Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента	*Кохана Євгенія Юрійовича*
	(ПІБ)
академічної групи	*122м-21-2*
	(шифр)
спеціальності	*122 Комп'ютерні науки*
	(код і назва спеціальності)
освітньої програми	*Комп'ютерні науки*
на тему:	*Розробка та дослідження ефективності роботи чат-бота для месенджеру Telegram на базі SQLite*

_____ *Є.Ю. Кохан*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	*Проф. Слєсарєв В.В.*			
Рецензент	*Доц. Кожевніков А.В.*			
Нормоконтролер	*Проф. Лактіонов І.С.*			

Дніпро
2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи
магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122М-21-2* *Кохану Євгенію Юрійовичу*
(група) (прізвище та ініціали)

на тему:	*Розробка та дослідження ефективності роботи чат-бота для месенджеру Telegram на базі SQLite*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 31.10.2022 р. № 1200-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Актуальність роботи зумовлена достатньо високою популярністю месенджерів та ботів серед користувачів мережі Інтернет.

Об'єктом дослідження є процес інформаційної підтримки пошуку найближчих заходів у містах України.

Метою роботи є розробка Telegram бота для пошуку найближчих заходів у містах України.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Практичним значенням є надання користувачам можливості пошуку та вибору заходів, враховуючи їх вподобання.

Обгрунтовано підставу для розробки програмного рішення, було здійснено деталізацію мети та задач проекту, визначено функціональні можливості чат-бота. Для розробки чат-бота було обрано такі технологій та засоби як Python, в якості мови програмування, Sublime Text як середовище розробки.

На етапі проектування було побудовано діаграми, які показують

функціональні можливості чат-боту, а саме було побудовано діаграми процесу забезпечення пошуку заходів по містах України у нотаціях IDEF0, діаграму варіантів використання чат бота та модель бази даних.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз існуючих рішень та постановка задачі роботи	12.09.2022-30.09.2022
Аналіз засобів створення програмного забезпечення побудови діаграми процесу забезпечення пошуку заходів по містах України у нотаціях IDEF0, діаграми варіантів використання чат бота та модель бази даних.	01.10.2022-31.10.2022
Розробка програмного забезпечення та дослідження ефективності запропонованих рішень.	01.11.2022-10.12.2022

Завдання видав _____ *Слєсарєв В.В.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Кохан Є.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *10.09.2022 р.*

Термін подання до ЕК *20.12.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 57 стор., 26 рис., 2 таблиці, 2 додатка, 18 джерел.

Об’єктом дослідження є процес інформаційної підтримки пошуку найближчих заходів у містах України.

Предметом дослідження є процес функціонування чат-бота з пошуку найближчих заходів у містах України.

Актуальність роботи зумовлена достатньо високою популярністю месенджерів та ботів серед користувачів мережі Інтернет.

Метою роботи є розробка Telegram бота для пошуку найближчих заходів у містах України.

Для досягнення мети необхідно вирішити такі задачі:

1) аналіз обраної предметної області;
2) порівняння програмних продуктів-аналогів; вибір технологій і середовища розробки;
3) здійснення проектування чат-бота; розробка чат-бота;
4) тестування чат-бота;
5) оформлення супровідної документації.

Практичним значенням є надання користувачам можливості пошуку та вибору заходів, враховуючи їх вподобання.

СПИСОК КЛЮЧОВИХ СЛІВ: ЧАТ-БОТ, PYTHON, SQlITE, TELEGRAM, АЛГОРИТМ, ПЛАТФОРМА.

ABSTRACT

Explanatory note: 57 pages, 26 figures, 2 tables, 2 appendices, 18 sources.

The object of the study is the process of information support for finding the nearest events in the cities of Ukraine.

The subject of the study is the process of functioning of the chatbot for finding the nearest events in the cities of Ukraine.

The relevance of the work is determined by the fairly high popularity of messengers and bots among Internet users.

To achieve the goal, it is necessary to solve the following tasks:

1) analysis of the selected subject area;

2) comparison of similar software products; choice of technologies and development environment;

3) designing a chat bot; chatbot development;

4) testing the chatbot;

5) preparation of accompanying documentation.

The practical value is to enable users to find and choose activities based on their preferences.

LIST OF KEYWORDS: CHAT-BOT, PYTHON, SQlITE, TELEGRAM, ALGORITHM, PLATFORM.

ЗМІСТ

ВСТУП

Сучасний світ постійно розвивається у різних сферах. У тому числі зміни відбуваються і у сфері інформаційних технологій, де постійно з'являються нові ідеї, які впливають на повсякденне життя людей.

Обмін повідомленнями став одним із найпопулярніших засобів масової інформації в останні роки. На сьогоднішній день окрім додатків та комп'ютерних програм все більшої популярності набуває використання різноманітних чат-ботів.

Чат-бот – це спеціалізований додаток, що дозволяє користувачам взаємодіяти зі сторонніми сервісами через інтерфейс чату. Він є своєрідним помічником, який спілкується з користувачами через повідомлення і має безліч функцій. Так, за допомогою ботів можна шукати необхідну інформацію, бронювати житло, отримати консультацію, здобувати знання та багато іншого.

Кожного дня люди відвідують різноманітні заходи такі як концерти, вистави, виставки, шоу та інші. Часто у людей виникає потреба знайти найближчі заходи у рідному місті, спираючись на конкретні вподобання, місце та час проведення. Гарним рішенням для задоволення даної потреби є розробка чат-бота для пошуку заходів у містах України.

Актуальність роботи зумовлена достатньо високою популярністю месенджерів та ботів серед користувачів мережі Інтернет.

Метою роботи є розробка Telegram бота для пошуку найближчих заходів у містах України.

Для досягнення мети необхідно вирішити такі задачі:

1) аналіз обраної предметної області;
2) порівняння програмних продуктів-аналогів; вибір технологій і середовища розробки;
3) здійснення проектування чат-бота; розробка чат-бота;
4) тестування чат-бота;
5) оформлення супровідної документації.

Практичним значенням є надання користувачам можливості пошуку та вибору заходів, враховуючи їх вподобання.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ РОБОТИ

1.1 Обгрунтування потреби у створенні чат-ботів

Оскільки, інформаційні технології стрімко розвиваються користувач стає все більш вимогливим до якості роботи технологій та їх функціоналу. Сьогодні використання інформаційних технологій є невід'ємною частиною повсякденного життя. Люди використовують інформаційні технології для комунікації, автоматизації бізнес-процесів, розваг, отримання нової інформації. Крім цього, існує велика кількість різноманітних месенджерів, що прискорюють процес комунікації між людьми. Одним із найбільш популярних та зручних месенджерів є Telegram.

Telegram – безкоштовний месенджер для смартфонів і персональних комп'ютерів, який працює під управлінням всіх найбільш поширених на сьогоднішній день операційних систем. Telegram надає можливість обмінюватися не лише текстовими повідомленнями, але і різними мультимедійними файлами [1]. Месенджер Telegram використовують більше ніж 500 мільйонів користувачів по всьому світу. Його переваги полягають у шифруванні та підвищеній безпеці інформації. За результатами досліджень сервісу Statista.com за липень 2021 року Telegram входить до рейтингу найбільш популярних месенджерів світу (рис. 1.1) [2].

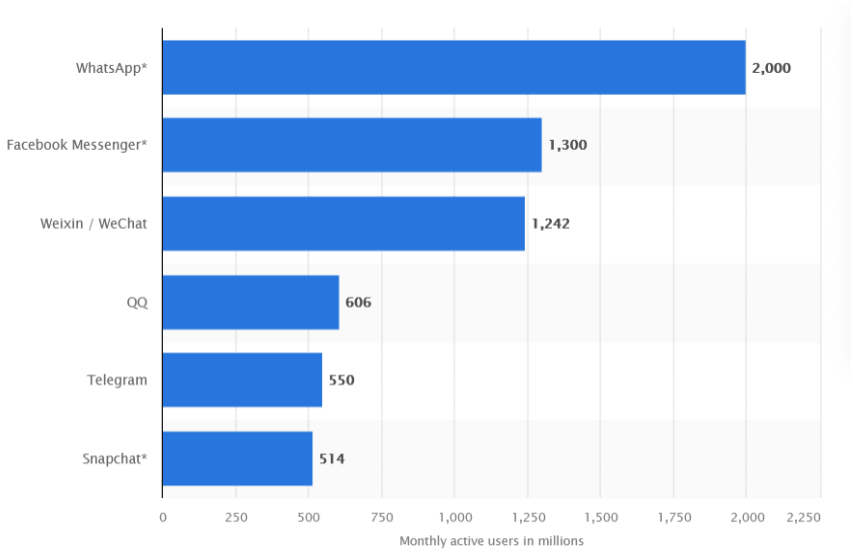

Рис. 1.1. Рейтинг найпопулярніших месенджерів світу

Також Telegram відомий своїми чат-ботами, які значно полегшують життя людини.

Чат-бот – це програма, яка імітує реальну розмову з користувачем. Чат-боти дозволяють спілкуватися за допомогою текстових або аудіо повідомлень на сайтах, в месенджерах та мобільних додатках [3]. Чат-боти в основному використовують штучний інтелект для спілкування з користувачами, тому надають релевантний контент і актуальні пропозиції. Вони функціонують на основі набору інструкцій або використовують машинне навчання. Чат-ботів вважають одним із найбільш досконалих та перспективних засобів організації взаємодії користувача з системою. До основних функцій чат-ботів належать:

1) підтримка клієнтів, яка здійснюється 24 години на добу;

2) клієнтський сервіс. За допомогою чат-бота можна робити покупки та запитувати різні послуги;

3) чат-боти допомагають оптимізувати в роботі такі процеси як: бронювання та замовлення квитків, інформування людей, розклад подій і багато іншого [4].

На даний момент існує велика різноманітність чат-ботів, які задовольняють різні потреби користувача:

1) чат-боти для бронювання квитків;
2) чат- боти для пошуку книг;
3) чат-боти для перегляду фільмів;
4) чат-боти для перегляду розкладу транспорту;
5) чат-боти для замовлення їжі;
6) чат-боти для бронювання номерів у готелях;
7) чат-боти для відстеження погодних умов;
8) чат-боти, які слугують для системної підтримки користувачів.

Незважаючи на те, що популярність у використанні ботів зростає, кількість чат-ботів, які б допомагали користувачам у пошуку найближчих заходів, відповіднодоїхрозташування та побажаньне висока. Том було прийнято рішення створити власного чат-бота, який би дозволив задовольнити потреби користувачів.

1.2 Опис технологій розоробки ботів

У 2016 році в галузі штучного інтелекту відбулося різке зростання інтересу до розумних співрозмовників, чат-ботів. Зростання обумовлено двома факторами: зростання попиту з боку бізнес-замовників та появою достатньої кількості інструментів та технологій для створення чат-ботів [5].

Чат-боти за принципом роботи можна розділити на два види: які працюють за заздалегідь заданим шаблоном і які навчаються в процесі спілкування. Шаблонні (скриптові) боти не розуміють природної мови. Діалог в таких ботах представляє заздалегідь формований шаблон, а скрипт – дерево рішень, в якому відповідь на питання відкриває новий, заздалегідь запрограмований сценарій. Діалоги в них зазвичай лінійні і структуровані. Шаблонні боти при спілкуванні з користувачем виділяють ключові слова і реагують на них. У таких ботах необхідно прописати команду для кожного слова чи фрази. Якщо при спілкуванні користувач не використовує ключові слова, то бот його не розуміє і виконує дії, передбачені для таких випадків, наприклад, пропонує звернутися до оператора [6].

Для чат-ботів такого типу використовуються технології базової обробки природної мови:

1) сегментація (розбиття на речення);
2) лематизація або стемінг (процес пошуку основи слова);
3) виділення ключових слів та/або іменованих сутностей;
4) І технології створення регулярних граматик (наприклад, AIML): регулярні вирази;
5) змінні та масиви для запам’ятовування контексту;
6) умови, цикли, рекурсії та ін.

Боти, які навчаються, розробляються на основі рішень штучного інтелекту і використовують при побудові діалогу оброблення природної мови і машинне навчання [7]. Принцип їх роботи заснований на аналізі діалогу для подальшого удосконалення своїх комунікативних навичок. Такі боти вміють оброблювати природну мову і коректно відповідати на

поставлені запитання [8]. Вони можуть ставити уточнюючі питання, щоб дізнатися про мету пошуку, і пропонують товар, виходячи з відповідей покупця. Чат-боти, що навчаються зберігають унікальні пошукові запити від різних користувачів і вдосконалюються, відповідаючи все більш точно.

Наявність великої кількості готових бібліотек спрощує роботу розробника при написанні бота. Завдання, які потрібно вирішувати програмісту, можна узагальнити невеликим списком:

1) реалізація розуміння роботом мови;
2) створення алгоритмів пошуку оптимальної відповіді;
3) інтеграція зі сторонніми API.

Написання інших алгоритмів для обробки та видачі даних.

У результаті аналізу технологій створення бота було визначено, що реалізація власного продукту буде здійснюватися за допомогою заздалегідь запрограмованого сценарію, оскільки розроблюваний бот повинен буде виконувати шаблонні команди. Також існує велика кількість різноманітних інструментів та бібліотек, що полегшують розробку бота.

1.3Аналіз існуючих ботів для пошуку найближчих заходів

Перед початком проектування та розробки чат-бота потрібно визначити та описати основні вимоги. Також для то55го, щоб переконатися в необхідності розробки потрібно проаналізувати предметну область, розглянувши вже існуючі рішення.

У ході дослідження було виділено такі аналоги:

1. Kyiv City Bot – бот який шукає та рекомендує івенти у місті Київ. Пошук відбувається серед різноманітності концертів, вечірок, лекцій, кінопоказів, вистав та інших заходів, які проходять у місті. Користувач має змогу обрати захід за кількома параметрами:
   - ✓ Місце;
   - ✓ Час

Kyiv City Bot простий у використанні і максимально швидкий, щоб користувачі з легкістю і не втрачаючи часу шукали улюблені заходи. Недоліками даного чат-бота є те що він шукає заходи лише у межах міста Київ, а також немає можливості пошуку подій за їх видом [9]. Приклад інтерфейсу чат-бота зображено на рисунках 1.2.

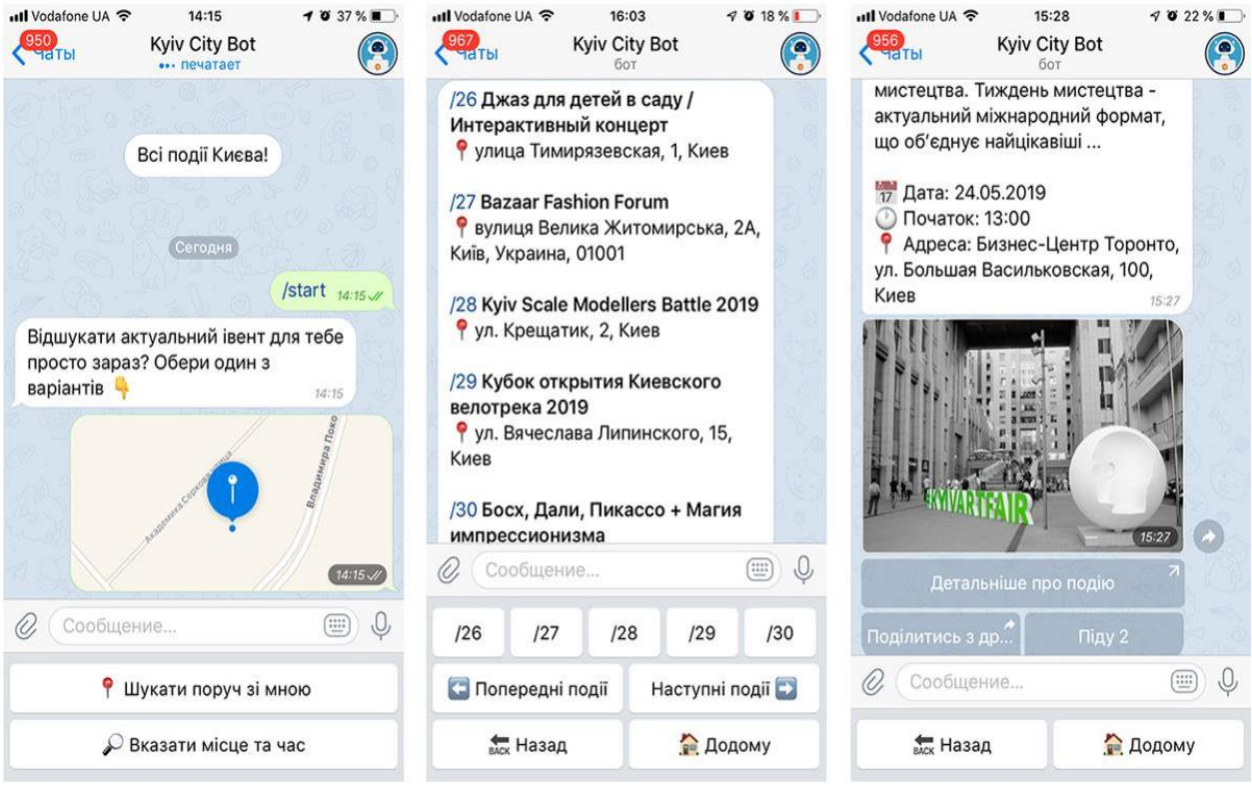

Рис. 1.2. Інтерфейс чат-бота Kyiv City Bot

MoeMisto.ua – бот, який надає можливість користувачеві отримувати інформацію про всі заходи у місті. Можливості бота містять пошук та підбір анонсів, подій та заходів по відповідним параметрам, а саме:

a. пошук анонсів по даті проведення заходу;
b. підбір анонсів по тематичним рубрикам, таким як концерти, вечірки, дитячі заходи, кіно, подорожі;
c. афіші розваг та відпочинку у закладах чи по місцях проведення.

Чат-бот надає можливість пошуку подій у таких містах України як: Вінниця, Житомир, Тернопіль, Одеса, Хмельницький, Київ, Львів, Кропивницький [10].

Приклад інтерфейсу бота MoeMisto.ua зображено на рисунку 1.3

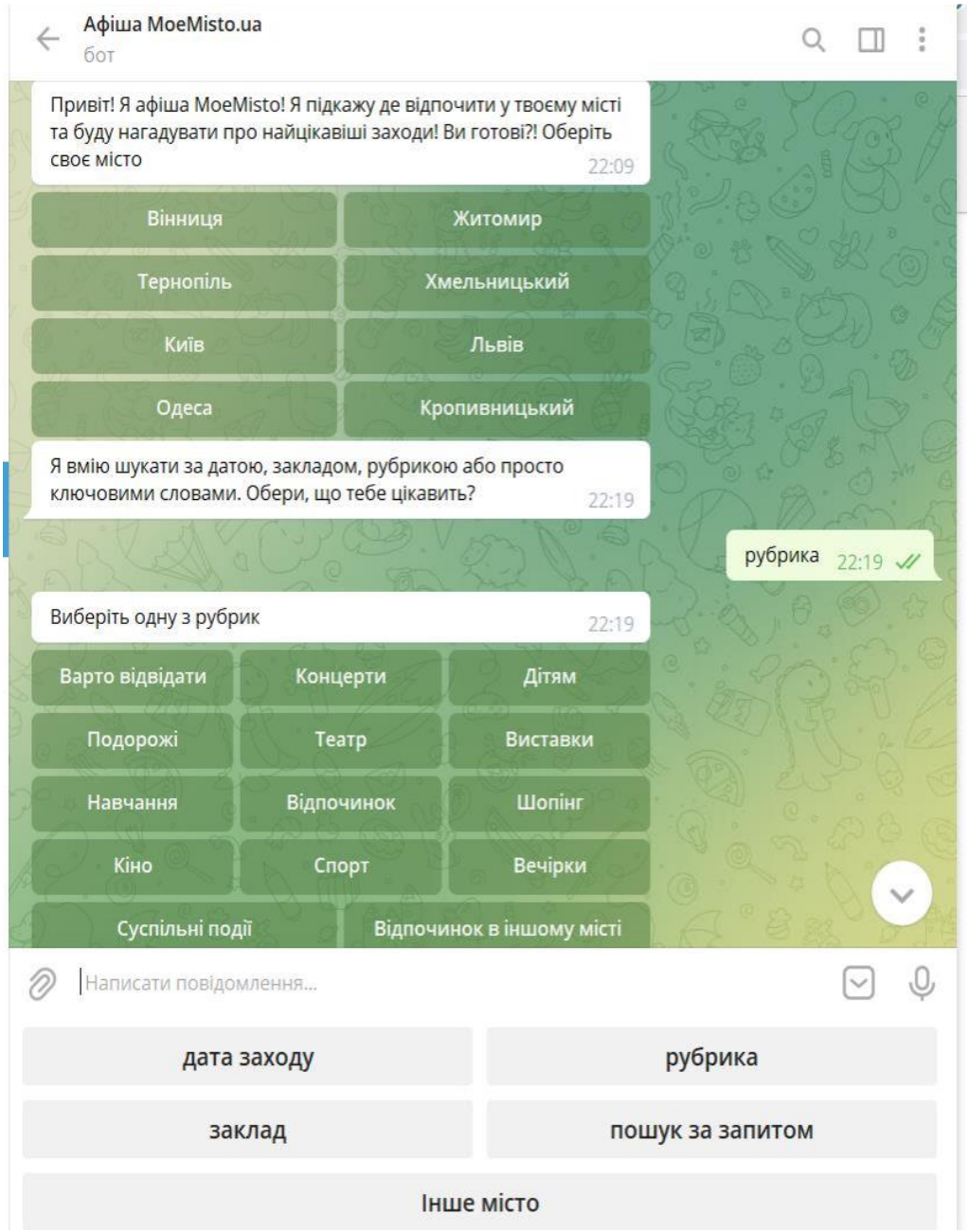

Рис. 1.3. Приклад інтерфейсу бота MoeMisto.ua

Music Trip Planner – бот для пошуку концертів музичних гуртів. Пошук здійснюється по назві гурту або місці проведення концерту [11]. На рисунку 1.4 зображено приклад інтерфейсу чат-бота.

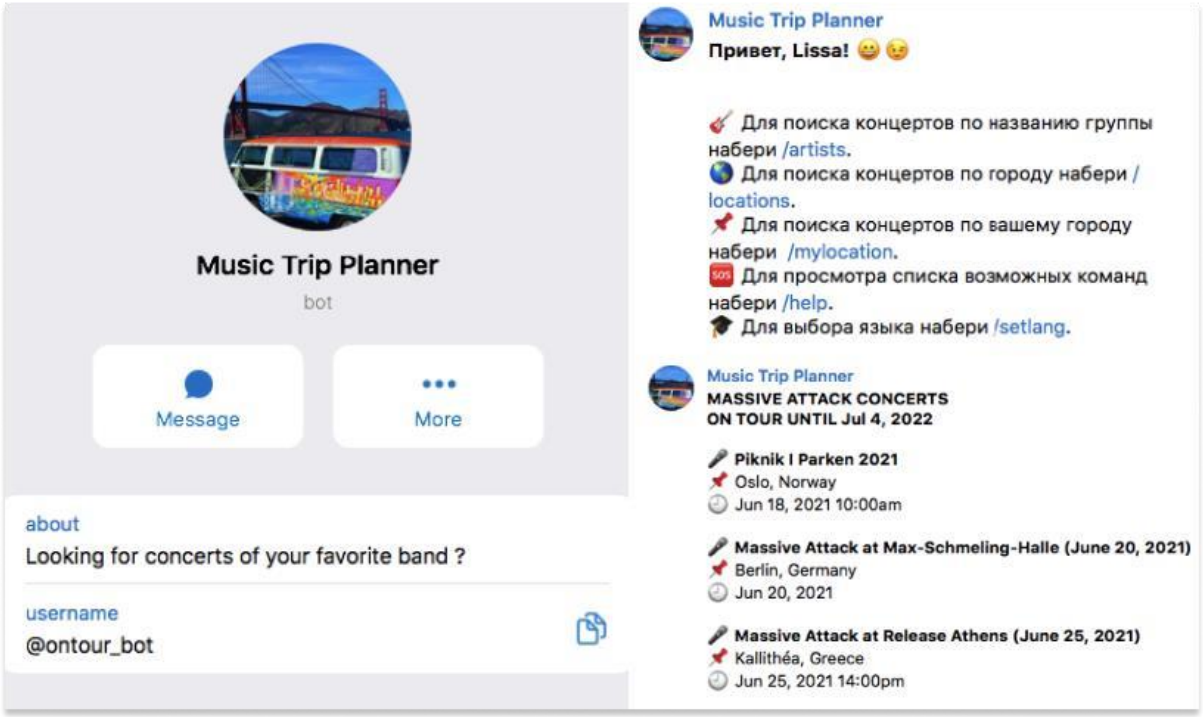

Рис. 1.4. Інтерфейс Music Trip Planner бота

Отже, в результаті аналізу аналогів було визначено, що для задоволення потреб користувача необхідно розробити чат-бот, що матиме такі функції: пошук заходів по місцю проведення, пошук за типом заходу, пошук заходів за датою, можливість зміни мови, перегляд детальної інформації про події.

1.4 Постановка задачі

Головною метою роботи є розробка Telegram бота для пошуку найближчих заходів у містах України. Даний проект буде мати цінність для великої кількості користувачів, які бажають знайти заходи у їхньому рідному місті.

Для досягнення мети роботи були визначені такі задачі:

1. здійснити аналіз предметної області;
2. здійснити аналіз вимог;
3. проаналізувати технології розробки ботів;
4. обрати методи та засоби реалізації;
5. здійснити проектування бота;

6. розробка бота;
7. тестування бота;
8. розробка супровідної документації.

Після встановлення мети та задач проекту було визначено, що чат-бот повинен мати такі функціональні можливості:

a) мати команди для інтерактивного спілкування з ботом;
b) мати команди для запуску роботи;
c) мати можливість пошуку заходів за місцем проведення;
d) мати можливість пошуку заходів за їх видами;
e) мати можливість пошуку заходів за датою проведення;
f) мати можливість пошуку заходів за ключовими словами;
g) мати можливість перегляду детальної інформації про захід;
h) мати можливість зміни мови інтерфейсу бота;
i) мати можливість встановлювати нагадування про наближення вибраних заходів.
j) мати можливість збереження заходів до списку бажань;
k) мати можливість фільтрації заходів.

Після виконання аналізу було встановлено що, чат-бот буде корисним для широкого кола користувачів для пошуку заходів за різними категоріями.

РОЗДІЛ 2
МЕТОДИ ТА ТЕХНОЛОГІЇ ВИРІШЕННЯ ЗАДАЧІ

2.1 Фреймворк React.js

Фреймворк – це каркас програми, який складається з набору бібліотек.

Бібліотека – код, який вбудовується у додаток та вирішує обмежений набір завдань залежно від того, які обов'язки на неї поклали розробники. Як правило, бібліотека – контейнер для набору класів та/або функцій, які розробники можуть використовувати у своєму додатку. [10]

Разом вони дають можливість користувачеві побудувати свою функціональність на певному фундаменті. Фреймворк задає структуру та підхід до архітектури програми, він диктує, як повинен писатися код.

Додаток може бути побудовано як тільки на одних бібліотеках, так і на одному фреймворку. Але частіше всього розробники використовують фреймворк та сторонні бібліотеки одночасно.

Фреймворки дозволяють створювати унікальний набір функцій і підходять для нетипових продуктів. JavaScript-фреймворки підвищують продуктивність розробника за рахунок того, що вже містять у собі певні правила, шаблони та набори функцій. [10]

Вибір того, який каркас взяти або з яких блоків побудувати новий проект, залежить від кругозору інструментів і поставлених задач.

Сучасні фреймворки достатньо функціональні та дозволяють розробникам реалізувати будь-яке завдання. Їх вибір досить широкий, але під час роботи над проектом були розглянуто 3 найпопулярніші – React, Vue та Angular.

У 2010 році Google випустив Angular і фреймворк зробив революцію, розробники його оцінили. Але продуктивність на той момент залишала бажати кращого. У 2016 році вийшов Angular2, написаний з нуля на TypeScript. Ця версія мала цілковито іншу архітектуру, більш низький поріг входження та якісну документацію з великою кількістю прикладів. Серед відомих компаній, що використовують цей фреймворк - Google, Forbes,

PayPal, Upwork та ін.

Майже паралельно з Angular розвивався React. Вперше його використали у новинній стрічці Facebook у 2011 році, але вихідний код відкрили лише у 2013 році. Поступово він набрав хорошу базу, велику кількість інструментів та широку підтримку ком'юніті. React використовується для розробки інтерфейсів у багатьох відомих компаніях: Facebook, Instagram, Netflix, BBC та ін.

2014 року з'явився Vue.JS. Біля його витоків стояв колишній співробітник Google . У фреймворку немає підтримки зі сторони великих компаній на відміну від React та Angular. Але це не завадило третій версії Vue розміщуватися у власному репозиторії GitHub і перейти на TypeScript. Це забезпечило їм більш чисту та зручну архітектуру вихідного коду, а також ряд інших змін. Vue використовується, в основному, китайськими гігантами - Alibaba, Tencent, Xiaomi, а також іншими великими компаніями.

Якщо звернутися до статистики числа завантажень (рис. 2.1), то React обирають частіше, ніж решту два фреймворки.

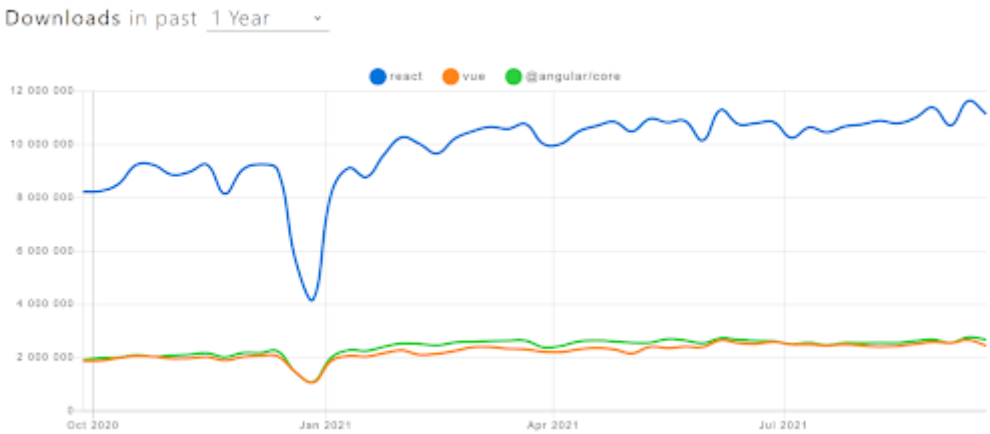

Рис. 2.1. Статистика числа завантажень різних фреймворків

Але за кількістю зірок (рейтинг затребуваності), які отримують репозиторії GitHub цих фреймворків, лідирує Vue. Хоча він і не набагато випереджає React. Рейтинг GitHub представлений на рис. 2.2.

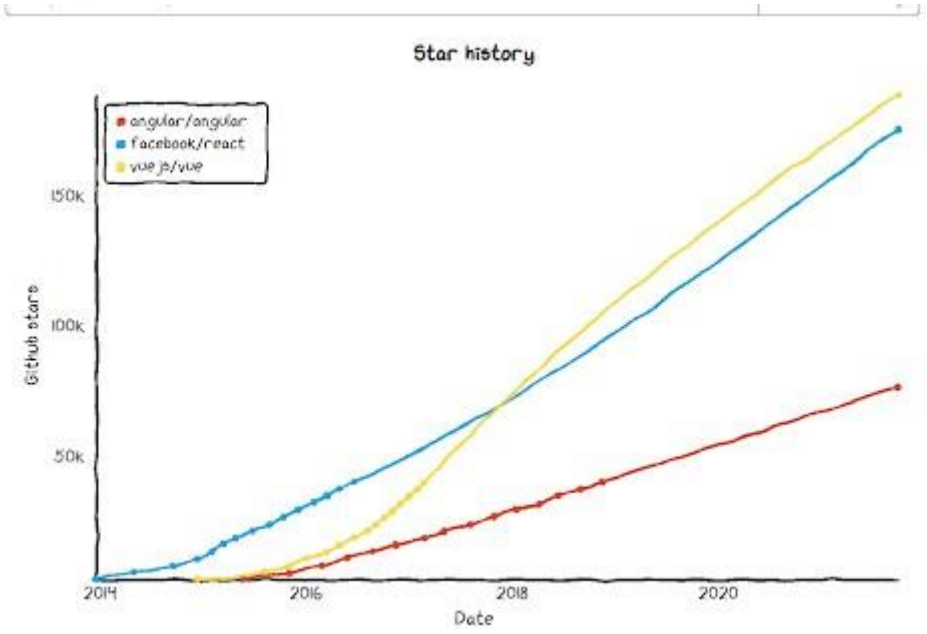

Рис. 2.2. Рейтинг затребуваності фреймоворків на веб-сервісі для розробників GitHub

Angular як фреймворк використовується для розробки великих корпоративних додатків, оскільки ряд архітектурних рішень йде з коробки, включаючи TypeScript. Це, з одного боку, може бути обмеженням, а з іншого – швидким рішенням стандартного кейсу. Також є механізми DI для зручності тестування та підміни залежностей. Angular як фреймворк вузькоспеціалізований. І поріг входження у розробку у ньому досить-таки високий. Крім знань JS-розробника, необхідно добре розуміти, як влаштований цей фреймворк. Можливо, саме тому він не користується особливою популярністю.

Vue підходить для середніх додатків та MVP. Висока швидкість на старті дозволяє зробити MVP за короткі терміни. Масштабованість програми Vue допомагає також динамічно розвивати проект. У версії 3.0 є повноцінна підтримка TypeScript. Легкість масштабування вища, ніж у Angular.

React, по своїй суті, є бібліотекою і має великий арсенал суміжних бібліотек. Він підходить для вирішення будь-яких задач. Його можна застосувати в MVP, невеликих та середніх проектах. Підтримка TypeScript допоможе в enterprise-розробках. Також його можна вибрати для розробки сайтів з мінімальною бізнес-логікою на фронті, оскільки є бойлерплейт

(шаблонний код).

Але при цьому у React немає «нав'язаних» рішень з коробки, тому потрібні хороший досвід розробки та знання як архітектурних патернів, так і роботи нативного JS.

Виходячи з того, що для роботи проектованої навчальної веб-платформи буде потрібна висока продуктивність, а реалізація всього передбаченого на платформі функціоналу потребуватиме широкого спектру інструментів та можливостей – було прийнято рішення використовувати фреймворк React.js. React не відноситься до фреймворків у чистому вигляді. Це свого роду модифікована бібліотека, «заточена» під MVC (Model-View-Controller, де Модель відповідає за надання даних, Вид – відповідає за відображення даних Моделі користувачеві, а Контролер інтерпретує дії користувача та змушує Модель вносити зміни).

React – бібліотека для відображення даних, яка включає в себе Virtual DOM, JSX, ізоморфність та компонентний підхід. [9]

React забезпечує підвищену гнучкість завдяк використанню «компонентів» – коротких ізольованих ділянок коду, які допомагають розробникам створювати складну логіку та UI. React взаємодіє з HTML через virtual DOM – копію реального DOM-дерева елементів сторінки. У копії всі елементи представлені як об'єкти JavaScript. Ці елементи, разом із декларативним стилем програмування React та одностороннім зв'язуванням даних, спрощують та прискорюють розробку. [9]

Віртуальний DOM – об'єкт, в якому зберігається інформація про стан інтерфейсу. Якщо стан змынюэться, наприклад, выдбувається відправка форми або натискається кнопка, React робить розрахунок різниці між попереднім та новим станами. Далі бібліотека відображає новий стан. Використання віртуального DOM дозволяє бібліотеці ефективно оновлювати реальний DOM.

React відрізняється від інших фреймворків та бібліотек тим, що не має вбудованого архітектурного шаблону. Він використовує компонентно-орієнтовану архітектуру. Користувальницькі інтерфейси React рендеруються компонентами, які працюють як функції та реагують на зміну даних. Таким

чином, внутрішня архітектура - постійна взаємодія між станом компонентів та діями користувачів. [9]

Одна з головних переваг React – легка масштабованість. Оскільки програми React використовують виключно JavaScript, розробники можуть застосовувати традиційні методи організації коду. Можливість багаторазового використання компонентів підвищує масштабованість додатків на даному фреймворку.

Ще одна перевага фреймворку – повна сумісність між версіями. Можливе підключення до додатку бібліотек різних версій, оновлення застарілих з успадкуванням властивостей.

За допомогою React можливо створювати веб-програми, які змінюють відображення без перезавантаження сторінки. Завдяки цьому програми можуть з високою швидкістю реагувати на дії користувачів, такі як заповнені форми, застосовані фільтри, додані у кошик товари тощо.

React застосовують для відображення компонентів інтерфейсу користувача. Також бібліотека може повністю управляти фронтендом. У цьому випадку React використовують разом з бібліотеками для керування станом і роутингу, наприклад, Redux і React Router.

Одна з ключових переваг React – універсальність. Цю бібліотеку можна використовувати на сервері та на мобільних платформах за допомогою React Native. Це принцип Learn Once , Write Anywhere або «Навчіться один раз, пишіть де завгодно».

React заснований на компонентах, це ще одна ключова особливість бібліотеки. Кожен компонент повертає частину користувальницького інтерфейсу зі своїм станом. Виконуючи об'єднання компонентів, розробник може створити завершений інтерфейс веб-програми.

Черговий плюс фреймворку – використання JSX. Це розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу. JSX схожий на HTML, проте це все-таки JavaScript. [11]

Таким чином, можна зробити висновок, що фреймворк React.js оптимально підходить для розробки проектованої у цій роботі навчальної веб-платформи. Використовуючи його, можна легко створювати складні

великомасштабні динамічні додатки, будувати користувальницькі інтерфейси. React.js забезпечує високий рівень продуктивності, має величезний вибір гнучких бібліотек та інструментів. Саме тому він був обраний для реалізації практичної частини даної роботи.

2.2 Вибір середовища розробки WebStorm

WebStorm – це інтегрована середовище для розробки на JavaScript та пов'язаних з ним технологіях. WebStorm дозволяє автоматизувати рутинну роботу і легко справлятися з складними завданнями, роблячи розробку більш простою і зручною. [12]

IDE створено на базі платформи IntelliJ, розробленої JetBrains та випущеної під відкритої ліцензією.

У WebStorm є вбудована підтримка множини технологій: JavaScript, TypeScript, React, React Native, Electron, Vue, Angular, Node.js, HTML, CSS та інших. Можна відразу приступати до написання коду, не витрачаючи час на встановлення та налаштування плагінів.

У WebStorm наявний розумний редактор коду. IDE добре розуміє структуру проектів та допомагає з усіма аспектами написання коду:

- Автодоповнення коду – дозволяє писати код швидше, використовуючи пропоновані IDE на ходу ключові слова та символи. Усі підказки формуються з урахуванням контексту та типів, а також працюють для різних мов. Наприклад, можна побачити підказки для імен класів з CSS під час роботи з .js файлами. Приклад автодоповнення коду представлений на рис. 2.3;
- Аналіз якості коду – WebStorm дозволяє з легкістю виявляти помилки у коді. IDE включає сотні інспекцій для всіх підтримуваних мов, а також перевірку правопису. Також вона інтегрується зі Stylelint, ESLint та іншими лінтерами – WebStorm запускає їх, поки пишеться код, і підсвічує помилки прямо у редакторі. IDE пропонує багато варіантів швидких виправлень;

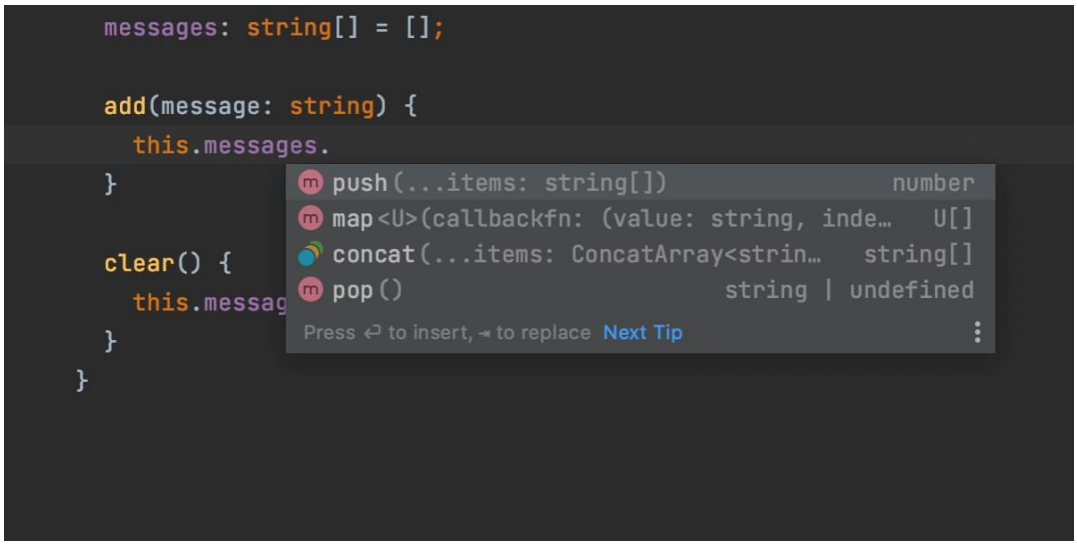

Рис. 2.3. Приклад автодоповнення коду в IDE WebStorm

- Безпечні рефакторинги – WebStorm включає потужні інструменти для рефакторингу коду по всій кодовий базі. Є можливість перейменовувати файли, папки та символи, отримувати компоненти, методи та змінні і не боятися, що ці дії приведуть до помилок. Редактор повідомляє про будь-які потенційні проблеми;
- Швидкий перегляд документації – якщо виникає необхідність подивитися документацію для символу, для цього можна не виходити з IDE. Достатньо навести курсор миші на потрібний символ, і після натискання комбінації клавіш Ctrl+Q можна переглянути відповідну інформацію (див. рис. 2.4). Також WebStorm підказує параметри для методів та функцій, які викликаються;

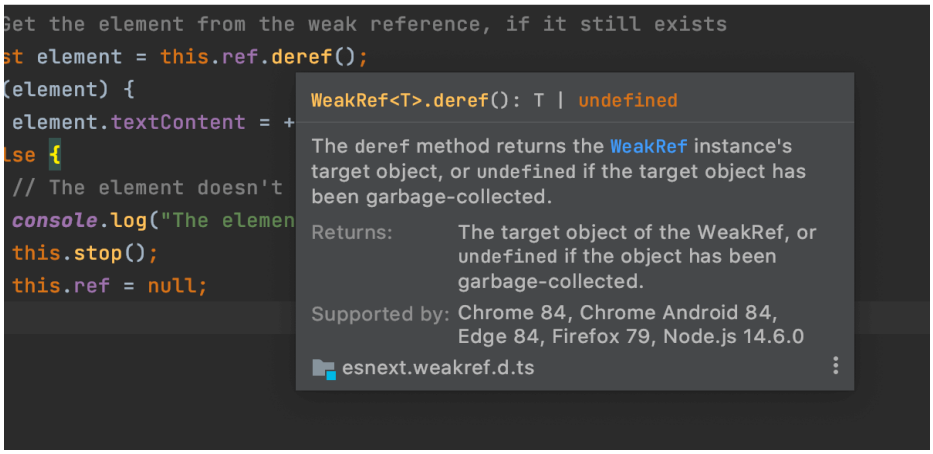

Рис. 2.4. Швидкий перегляд документації в IDE WebStorm

- Попередній перегляд HTML-файлів – у WebStorm є прев’ю для статичних HTML-файлів. Під час редагування HTML-коду або пов'язаних CSS та JavaScript-файлів, зміни зберігаються і прев'ю оновлюється автоматично.

Також доступні вбудовані інструменти для розробників. Один з головних плюсів IDE у тому, що вони об'єднують усі необхідні інструменти. WebStorm можна використовувати для налагодження та тестування клієнтського коду та програм на Node.js, а також для роботи з системою контролю версій. При розробці зручно користуватися: [12]

- Налагодженням JavaScript – запуск та налагодження коду клієнтських додатків та додатків на Node.js можна робити прямо у редакторі. Є можливість розставляти точки зупинки, виконувати програму покроково, додавати watches і т. д. – все це працює для самих різних типів додатків, включаючи JavaScript, TypeScript та Vue;
- Юніт-тестуванням – передбачена можливість писати, запускати та налагоджувати юніт-тести за допомогою Jest, Mocha, Karma, Protractor та Cucumber.js. Переглядати результати тестів можна у вигляді дерева, що дозволяє переходити до вихідного коду тесту;
- Просунутою інтеграцією з VCS – інтерфейс WebStorm підтримує безліч повсякденних операцій: порівняння гілок, перегляд та вирішення конфліктів і т. д. IDE також дозволяє працювати з проектами, розміщеними на GitHub;
- Локальною історією змін – на випадок, якщо при розробці не було зроблено коміт або були випадково видалені кілька файлів, у WebStorm є вбудована локальна історія. Вона відстежує всі зміни в рамках проекту та дозволяє скасувати їх, навіть якщо проект поки що не підключений до системи контролю версій; [12]
- Вбудованим HTTP- клієнтом – є можливість тестувати веб-сервіси за допомогою вбудованого HTTP-клієнта – створювати, редагувати та виконувати HTTP- запити прямо в редакторі;
- Підтримкою лінтерів – доступна можливість інтеграції WebStorm з популярними лінтерами, такими як ESLint, Stylelint або TSLint. Це

дозволить переглядати згенеровані ними попередження та помилки прямо в IDE і швидко виправляти проблемний код;

- Вбудованим терміналом – можливе використання вбудованого терміналу для роботи з будь-якою командною оболонкою прямо з IDE.

Ще одна перевага редактора – швидка навігація та пошук. За допомогою даної IDE можливо швидко переміщатися кодом незалежно від розмірів проекту. Доступний пошук файлів, класів або символів, є можливість переглядати всі результати в одному місці. [12]

Головні опції навігації та пошуку:

- Вспливаюче вікно Search Everywhere – допоможе знайти у WebStorm все що завгодно. Його можна використовувати для пошуку дій, файлів, класів або символів та перегляду всіх знайдених збігів;
- Дослідження коду – редактор дозволяє швидко перейти до оголошення символу і показує де він використовується в проекті. Також є можливість легко подивитися визначення символу, не переходячи до його оголошення;
- Навігація проектом – можна переглядати файли проекту, переходити до файлів та фрагментів коду, які недавно були відкриті або редагувалися – WebStorm запам'ятовує, з чим ведеться робота, і дозволяє швидко повернутися назад. Для навігації використовуються вкладки або поєднання клавіш. Вид вікна навігації представлений на рис. 2.5;

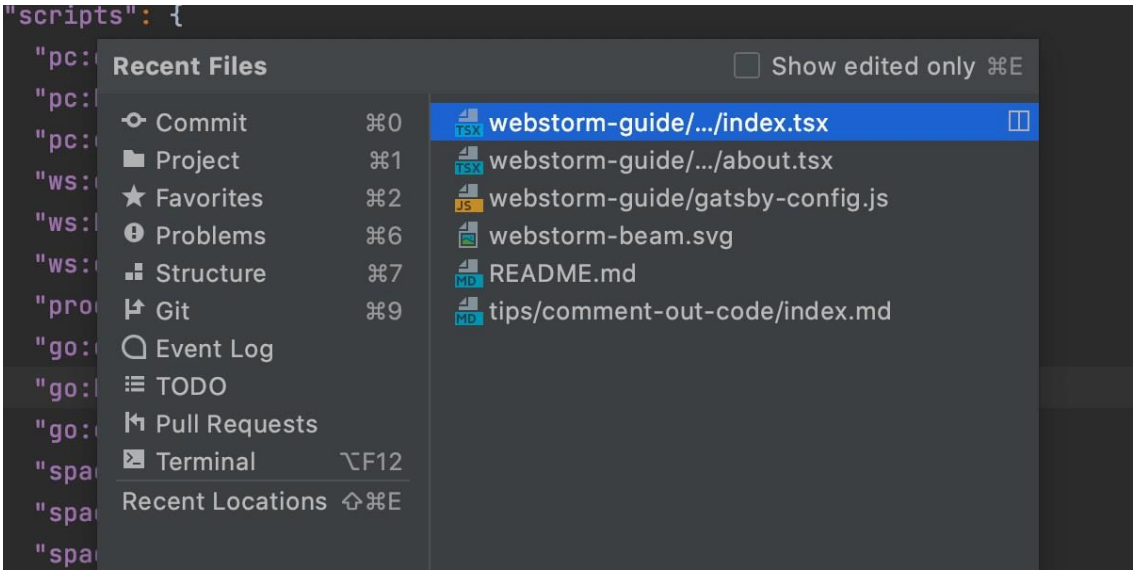

Рис. 2.5. Вікно навігації у IDE WebStorm

- Пошук тексту – IDE дозволяє знаходити та замінювати текстові рядки як у певних файлах, так і у всьому проекті. Можливо звузити область пошуку до виділеного фрагмента коду або використовувати різні області проекту та фільтри.

Ефективна командна робота теж є важловою складовою редактора. WebStorm дозволяє програмувати разом із командою в режимі реального часу та спілкуватися з колегами прямо з IDE. Головні можливісті:

- Віддалена спільна розробка – WebStorm включає Code With Me – сервіс для спільної віддаленої розробки та парного програмування. Його зручно використовувати, щоб разом працювати над кодом та спілкуватися з колегами, не залишаючи IDE;
- Можливості для розподілених команд – дозволяють створювати пул-реквести GitHub, об'єднувати їх з цільовою гілкою, робити код-рев'ю – для цього не потрібно залишати IDE. Можна настроїти інтеграцію з JetBrains Space, щоб переглядати та клонувати його репозиторії та виконувати код-рев'ю;
- Можливість ділитися налаштуваннями проекту – щоб дотримуватися єдиного стилю коду, можна поділитися з командою налаштуваннями стилю коду, використовуючи конфігурацію IDE, Prettier або EditorConfig. Також можна поділитися з колегами і іншими налаштуваннями проекту, наприклад, конфігураціями запуску.
- Інтеграція з баг-трекерами – передбачена можливість підключення WebStorm до баг-трекера та роботи над завданнями прямо з IDE.

Доступна можливість налаштовувати інтерфейс WebStorm під свої потреби, за допомогою різних тем і плагінів. Можливо також зберегти налаштування та використовувати їх в інших інсталяціях WebStorm. [12]

Для користувача доступні:

- Налаштування інтерфейсу – у WebStorm є кілька готових тем інтерфейсу, але можливо також виставити налаштування самостійно. Можна налаштувати видимість різних елементів інтерфейсу та поміняти їх розташування.
- Великий вибір розкладок – IDE пропонує поєднання клавіш

майже для будь-якої дії. Можна використовувати будь-яку з існуючих розкладок чи створити власну. Також є можливість встановити розкладки інших редакторів, включаючи Vim, VS Code та Sublime Text.

- Плагіни – доступна можливість розширювати базову функціональність IDE і поповнювати набір налаштувань, використовуючи колекцію плагінів (див. рис. 2.6).

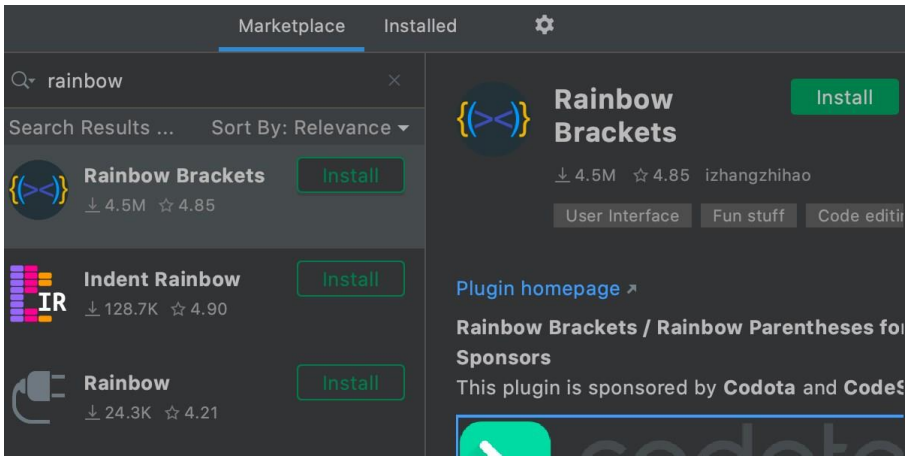

Рис. 2.6. Колекція плагінів IDE

Підтримка інструментів для роботи з базами даних та SQL (див. рис. 2.7)

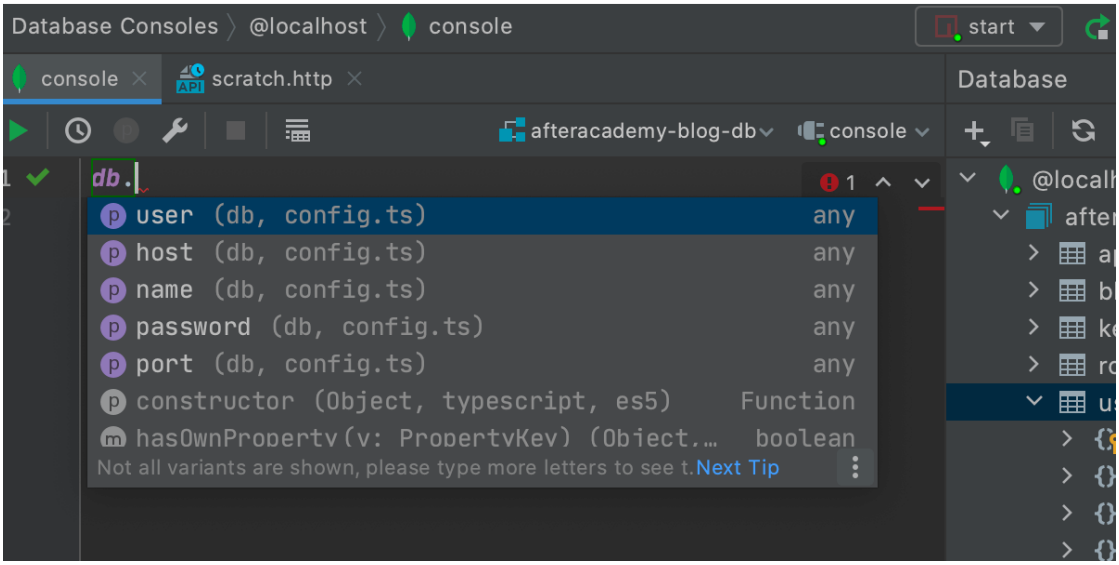

Рис. 2.7. Консоль для роботи з базами даних у IDE

Спеціальні можливості – редактор пропонує безліч спеціальних можливостей для різноманітних потреб. Можна налаштовувати кольори елементів інтерфейсу, розмір вікон та шрифтів у редакторі, коригувати колірну схему у разі порушеного кольоросприйняття, поєднання клавіш та

багато іншого.

- Режим швидкого редагування файлів - режим LightEdit дозволяє відкривати файл у полегшеному текстовому редакторі, що підтримує підсвічування синтаксису, автоматичне збереження та інші базові можливості.

Підводячи підсумок, можна сказати, що WebStorm – потужний редактор основних мов, який добре розпізнає структуру проекту, сканує ймовірні складності ще до відкриття розробки в браузері, рекомендуючи їх ефективне рішення. Продуктивності редактору надають включені IDE-інструменти для створення та тестування проектів. Редактор містить у собі масу зручних функцій для розробників. Виходячи з всього переліченого вище, було прийнято рішення використовувати для реалізації практичної частини проекту IDE WebStorm.

2.3. Вибір засобів реалізації мобільного чат-бота

Перед початком реалізації чат-бота потрібно обрати засоби реалізації. В якості мови програмування було обрано Python.

Python – це високорівнева мова програмування загального призначення, яка використовується навіть для розробки веб-додатків та чат-ботів. Мова орієнтована на підвищення продуктивності розробника і читання коду [12].

Python підтримує кілька парадигм програмування: структурне, об'єктно-орієнтоване, функціональне, імперативне та аспектно-орієнтоване. У мові є динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізм обробки винятків, підтримка багатопоточних обчислень та зручні високорівневі структури даних. Програмний код Python організовується в функції і класи, які можуть об'єднуватися в модулі, а вони в свою чергу можуть бути об'єднані в пакети. Python зазвичай використовується як інтерпретований, але також може бути скомпільований у байт-код Java та в MSIL (в рамках платформи .NET).

Python – одна з основних мов програмування, які застосовують у

галузі машинного навчання та штучного інтелекту (Machine Learning та Artificial Intelligence). Python займає друге місце у списку найпопулярніших мов програмування. Він випереджає JavaScript, PHP, Swift та інші поширені мови, поступаючись лише C.

У якості редактору коду було обрано Sublime Text. Sublime Text Editor – це повнофункціональний текстовий редактор для редагування локальних файлів або бази коду. Він включає різні функції для редагування бази коду, яка допомагає розробникам відстежувати зміни.

Sublime Text editor використовується як інтегрований редактор розробки (IDE), як Visual Studio і NetBeans. Поточна версія редактора Sublime Text – 3.0 та сумісна з різними операційними системами, такими як Windows, Linux та MacOS.

SQLite – це система управління базами даних, відмінною особливістю якої є її вбудовування в додатки. Використовуючи високоефективну інфраструктуру, SQLite може працювати в невеликому обсязі пам’яті, що виділяється для неї, набагато меншому, ніж у будь-яких інших системах БД. Це робить SQLite дуже зручним інструментом з можливістю використання практично в будь-яких завданнях, що покладаються на базу даних. До переваг SQLite відносяться: надійність, відкритість вихідних кодів, наявність консольної утиліти для роботи з базами, популярність [14].

Також для розробки бота для месенджеру Телеграм знадобиться пакет python-telegram-bot – оболонка для API від Telegram. Написати бота за допомогою цієї бібліотеки дуже просто, оскільки вона повністю сумісна з Python 3.6+ [15]. Також для створення чат-бота необхідний Telegram канал @BotFather, який відповідає за реєстрацію нових ботів. Він також відповідає за базове налаштування (опис, фото профілю, вбудована підтримка тощо).

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЙОГО ЗАСТОСУВАННЯ

3.1 Діаграми процесу забезпечення пошуку заходів по містах України у нотаціях IDEF0

Перед розробкою програмного продукту потрібно визначити структуру та функції чат-бота та побудувати функціональну модель. Функціональна модель призначена для опису існуючих бізнес-процесів, у яких використовуються як природна, так і графічна мови [16]. З цією метою було побудовано діаграми процесу забезпечення пошуку заходів по містах України у нотаціях IDEF0.

IDEF0 – методологія функціонального моделювання. За допомогою наочної графічної мови IDEF0, система, що вивчається, постає перед розробниками та аналітиками у вигляді набору взаємопов'язаних функцій (функціональних блоків – у нотаціях IDEF0). Як правило, моделювання засобами IDEF0 є першим етапом вивчення будь-якої системи [17].

Для відображення структурно-функціональної моделі процесів роботи чат-бота спочатку було побудовано контекстну діаграму, що відображає процес забезпечення пошуку заходів по містах України. Для її побудови було виділено: вхідні дані, вихідні дані, механізми та керуючі елементи.

Вхідними даними процесу є запит користувача.

Вихідними даними є:

a) результати пошуку заходів;
b) заходи, додані до списку «Обране»;
c) налаштовані нагадування про заходи;
d) нові додані заходи до БД.

Керуючими елементами процесу в даному випадку є:

a) функціональні вимоги до бота;
b) інструкції користувача.

Механізмами є:

a) мобільний чат-бот;
b) база даних;
c) Telegram.

Контекстну діаграму зображено на рисунку 3.1

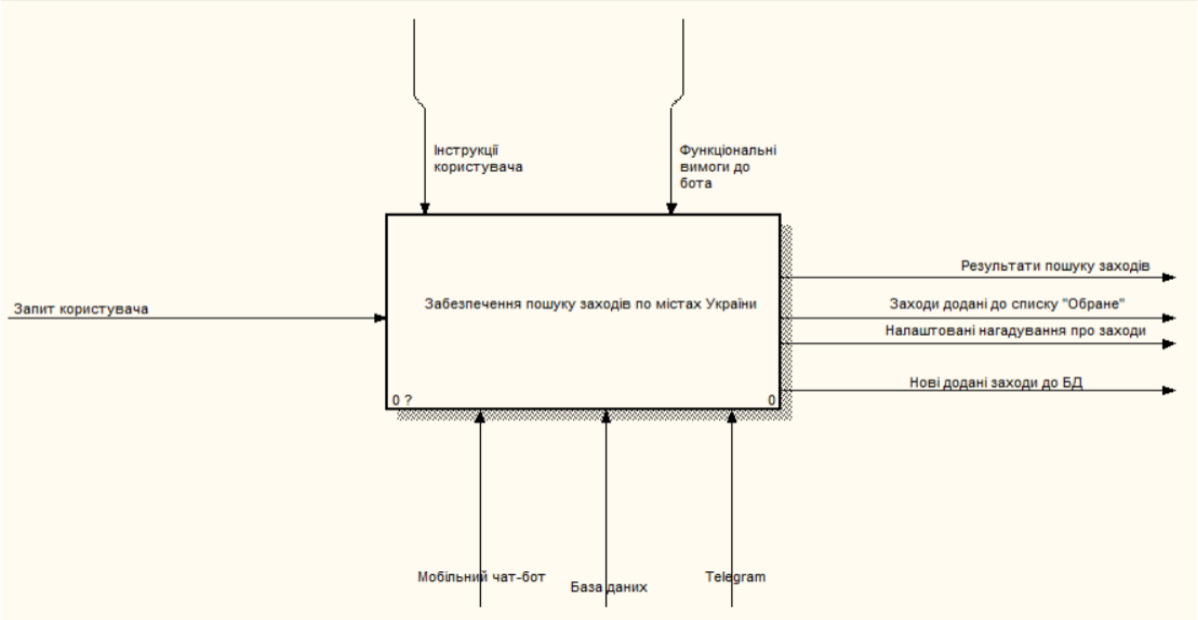

Рис. 3.1. Контекстна діаграма процесу забезпечення пошуку заходів по містах України

Після побудови контекстної діаграми потрібно здійснити функціональну декомпозицію. Для цього систему потрібно розбити на підсистеми і описати кожну з них окремо (діаграма декомпозиції).

Після декомпозиції процесу було виділено такі блоки:

1. авторизація користувача у системі;
2. вибір мови інтерфейсу;
3. додавання власних заходів;
4. пошук заходів;
5. додавання заходів до списку «Обрані»;
6. створення нагадувань про обраний захід.

Діаграму декомпозиції процесу зображено на рисунку 3.2

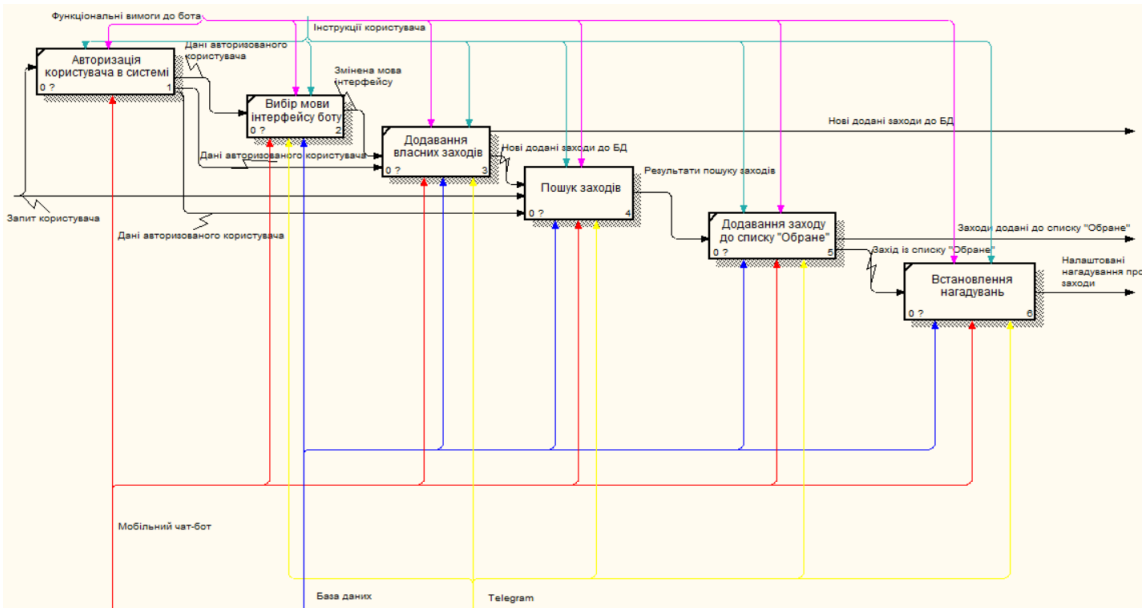

Рис. 3.2. Діаграма декомпозиції процесу забезпечення пошуку заходів по містах України

Далі було здійснено декомпозицію другого рівня, де було декомпозовано безпосередньо сам процес пошуку заходів.

У результаті декомпозиції процесу пошуку заходів було виділено такі блоки:

a) вибір міста;

b) вибір критеріїв для подальшого пошуку;

c) вибір категорії заходу;

d) обробка введених даних системою.

Діаграму декомпозиції другого рівня зображено на рисунку 3.3.

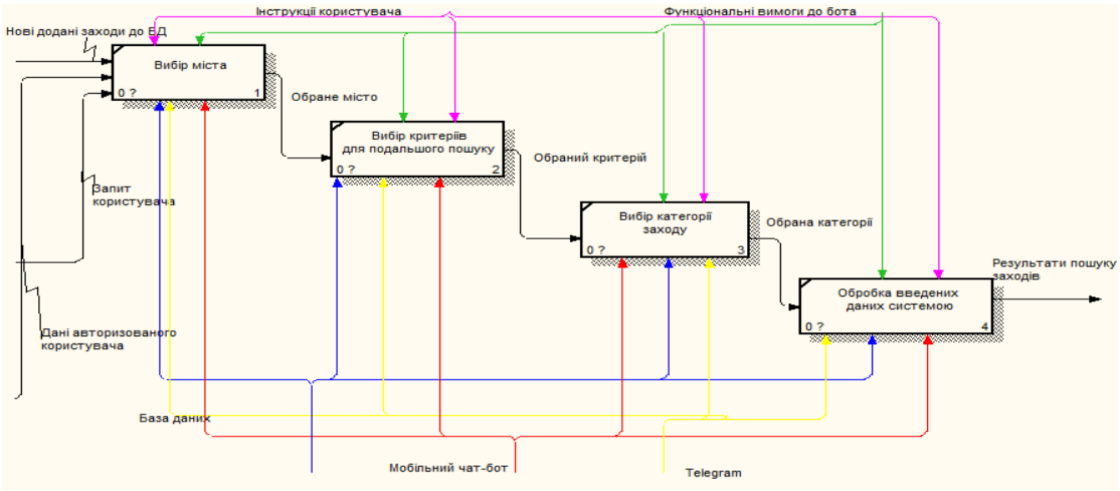

Рис. 3.3. Діаграма декомпозиції другого рівня

3.2 Моделювання варіантів використання чат-бота

На наступному етапі було створено діаграму варіантів використання чат-бота.

Діаграма варіантів використання – діаграма, яка описує, який функціонал розробленої програмної системи, доступний кожній групі користувачів [18].

На діаграмі використання зображаються:

a) актори – групи осіб або систем, що взаємодіють із розроблюваною системою;
b) варіанти використання (прецеденти) – послуги, які розроблювана система надає акторам;
c) коментарі;
d) відносини між елементами діаграми [19].

Діаграму варіантів використання зображено на рисунку 3.4.

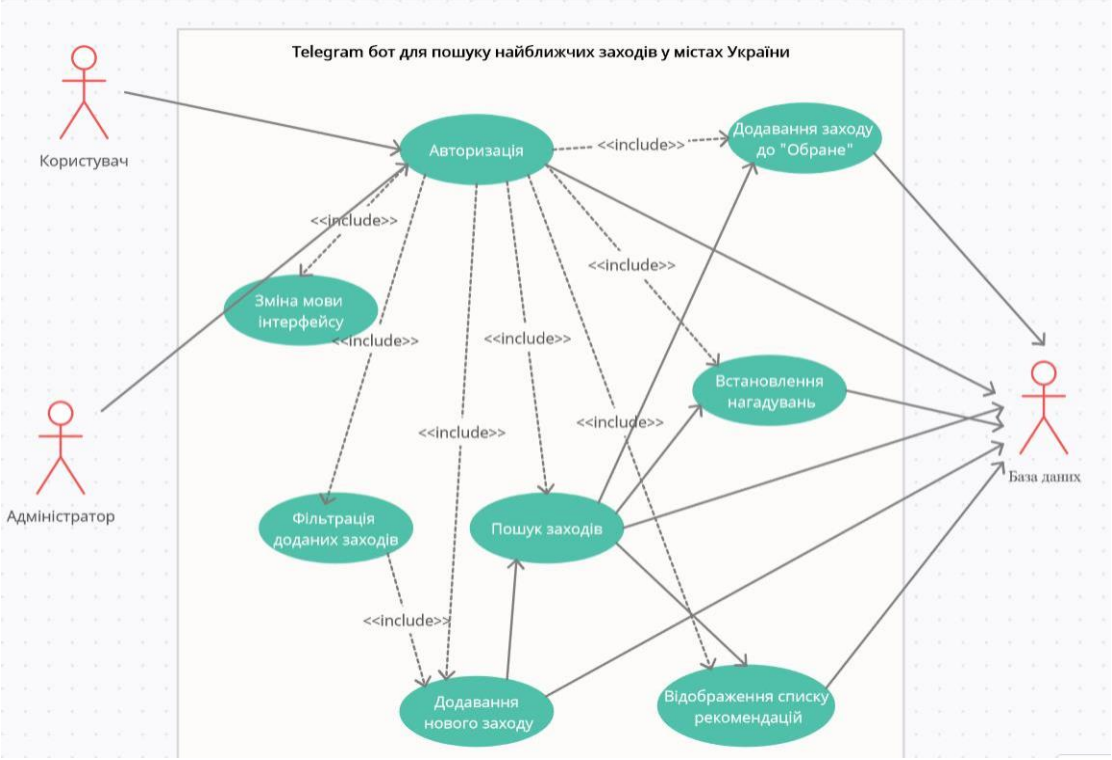

Рис. 3.4. Діаграма варіантів використання

За актора було виділено користувача чат-бота, який здійснює пошук заходів, адміністратора, який відповідає за фільтрацію заходів та базу даних, що зберігає дані, які використовуються або додаються впродовж роботи з ботом.

3.3 Проектування моделі бази даних

ER-діаграма – це різновид блок-схеми, де показано, як різні сутності пов’язані між собою всередині системи. ER-діаграми найчастіше застосовуються для проектування та налагодження реляційних баз даних у сфері освіти, дослідження та розробки програмного забезпечення та інформаційних систем для бізнесу [20]. ER-діаграми (або ER-моделі) покладаються на стандартний набір символів, включаючи прямокутники, ромби, овали та сполучні лінії для відображення сутностей, їх атрибутів та зв'язків. Ці діаграми влаштовані за тим самим принципом, як і граматичні структури [21].

Далі було здійснено проектування бази даних та було створено ER-діаграму.

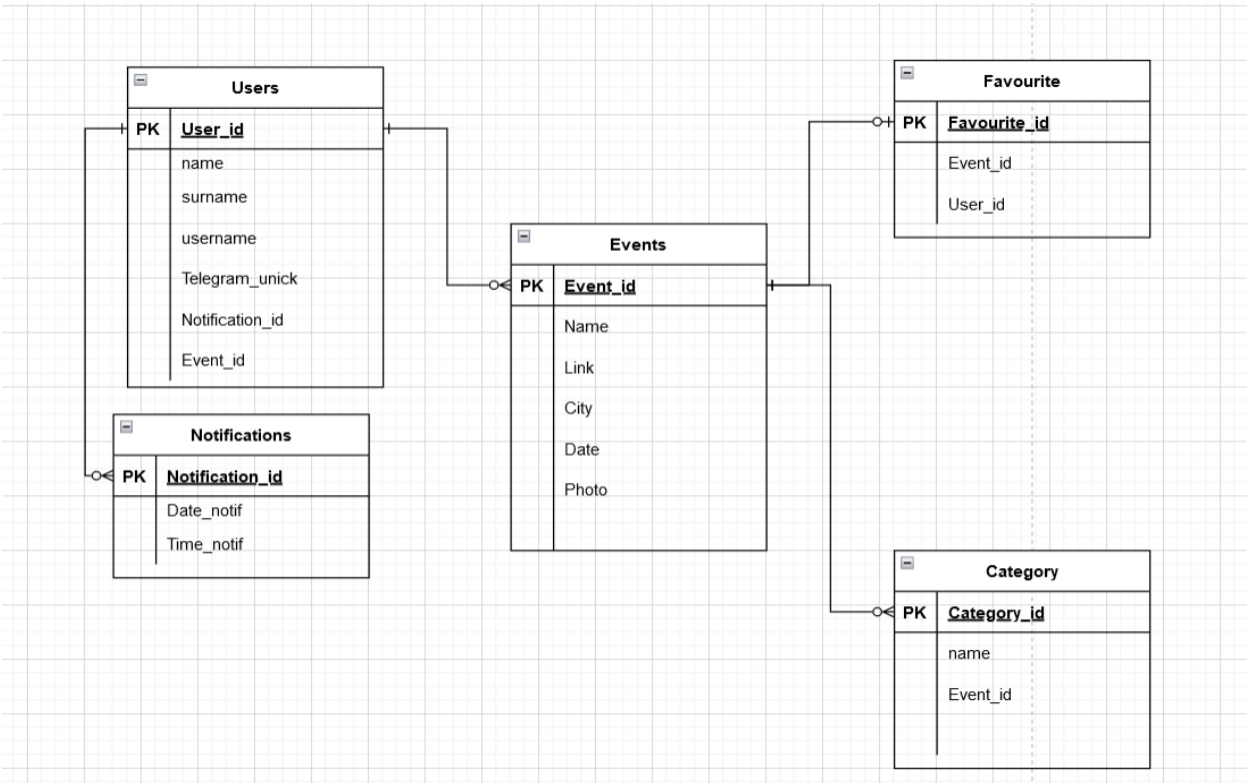

Рис. 3.5. ER-діаграма

Дана база даних містить 6 таблиць, що мають певні атрибути:

1. Users – таблиця, що містить дані про користувачів чат-бота. Таблиця містить такі атрибути: User_id (ідентифікатор користувача), name (ім’я користувача в месенджері Телеграм), surname (прізвище користувача в месенджері Телеграм) та username (нік користувача в месенджері Телеграм), Notification_id (ідентифікатор нагадування), Event_id

(ідентифікатор заходу);

2. Events – таблиця, що містить дані про існуючі заходи. Таблиця містить такі атрибути: Event_id (ідентифікатор заходу), Name (назва заходу), Photo (фото заходу), City (місце проведення), Date (дата заходу), Link (посилання);

3. Category – таблиця, що містить назви категорій заходів. Таблиця містить такі атрибути: Category_id (ідентифікатор категорії), Name (назва категорії), Event_id (ідентифікатор заходу);

4. Notifications – таблиця, що містить список із створеними нагадуваннями. Таблиця містить такі атрибути: Notification_id (ідентифікатор нагадування), Date_notif (дата нагадування), Time_notif (час нагадування);

5. Favourite – таблиця, що містить заходи із списку «Обране». Таблиця містить такі атрибути: Favourite_id (ідентифікатор обраного заходу), Event_id (ідентифікатор заходу), User_id (ідентифікатор користувача).

3.4. Архітектура чат-бота

Архітектура чат-бота представляє концепцію, яка визначає структуру і взаємозв'язки між його компонентами [22].

До компонентів архітектури чат-бота відносяться:

1) Telegram – месенджер, що є платформою для розробки чат-бота;
2) Telegram Bot API – є посередником між Telegram та обробником команд, саме він отримує команди, у вигляді повідомлень, від користувача, а також відправляє відповідь на ці команди користувачу;
3) Обробник команд отримує команди від Telegram Bot API і за допомогою прописаних сценаріїв та бази даних відправляє відповідь до Telegram Bot API.

Архітектура чат-бота зображена на рисунку 3.6

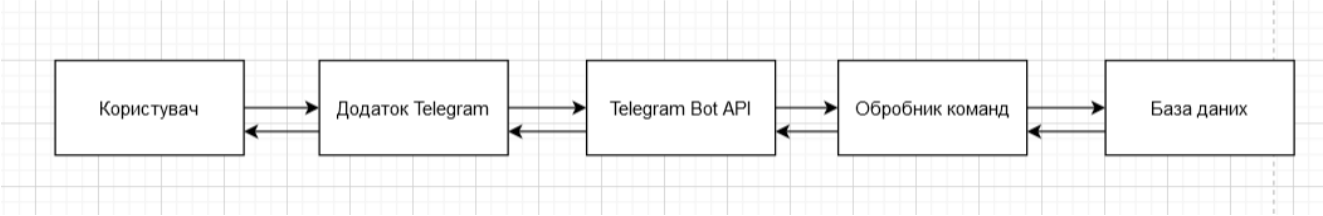

Рис. 3.6. Архітектура чат-бота

Перед початком розробки чат-бота необхідно необхідно зареєструвати його за допомогою спеціального бота @BotFather [23]. Реєстрацію чат-бота зображено на рисунку 3.7.

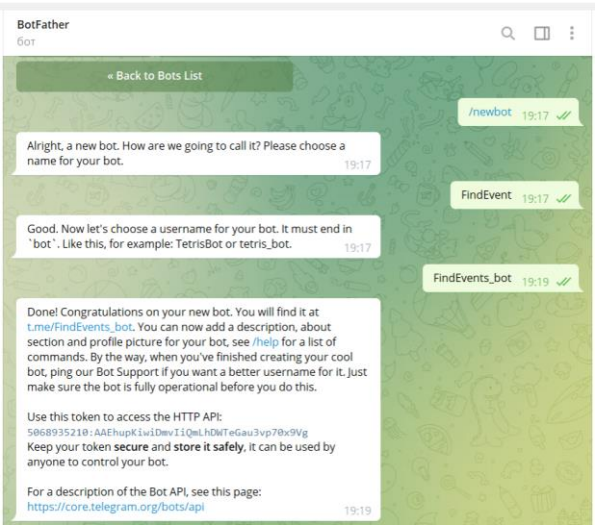

Рис. 3.7. Реєстрація бота

Далі було встановлено фото профілю та опис бота. Фото профілю та опис бота зображено на рисунку 3.8.

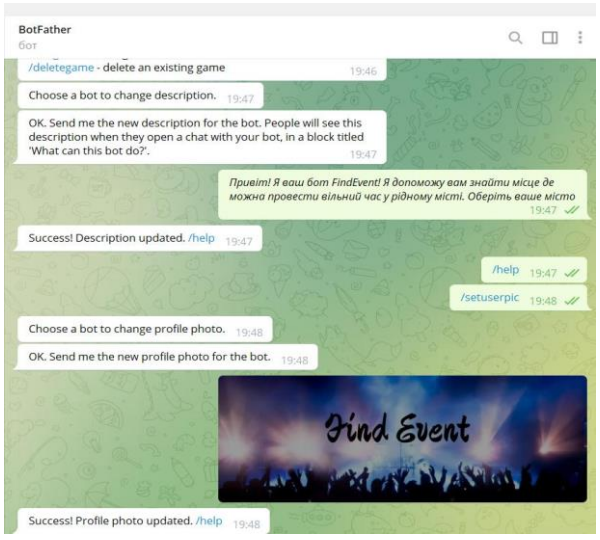

Рис. 3.8. Встановлення фото прфілю та опису бота

Наступним кроком було імпортовно необхідні бібліотеки та підключено токен отриманий при реєстрації бота [24].

Для розробки Telegram бота було підключено такі бібліотеки [25]:

– sqlite3;

– telebot;

– random.

Фрагмент даного коду зображено на рисунку 3.9.

```
#!/usr/bin/python
# coding: utf-8
import sqlite3
import telebot
import random
from keyboards import *

database = "database.db"
statebase = "state.db"

bot = telebot.TeleBot("5082705820:AAGTATaDZalIAXTsGQkUCmLbcbP1Ehipd0w")
```

Рис. 3.9. Імпорт бібліотек та підключення токену

Далі було створено обробник для команди Start [26]. Фрагмент коду обробника команди Start зображено на рисунку 3.10.

```
@bot.message_handler(commands=['start'])
def welcome(message):
    con = sqlite3.connect(database)
    cur = con.cursor()
    cur.execute("SELECT id FROM users WHERE id = ?",(message.chat.id,))
    user = cur.fetchone()
    if user == None:
        con = sqlite3.connect(database)
        cur = con.cursor()
        cur.execute(f"INSERT INTO users (id)"f"VALUES ({message.chat.id})")
        con.commit()
    bot.send_message(message.chat.id,"<i>Привіт! Я ваш бот FindEvent! Я допоможу вам знайти місце де можна провести вільний час у рідному місті. Оберіть ваше
```

Рис. 3.10. Обробник події Start

Для початку роботи з ботом користувачу необхідно ввести команду /start. Після цього відбудеться запуск бота та з’явиться коротке повідомлення та список міст для пошуку заходів. Процес запуску бота зображено на рисунку 3.11.

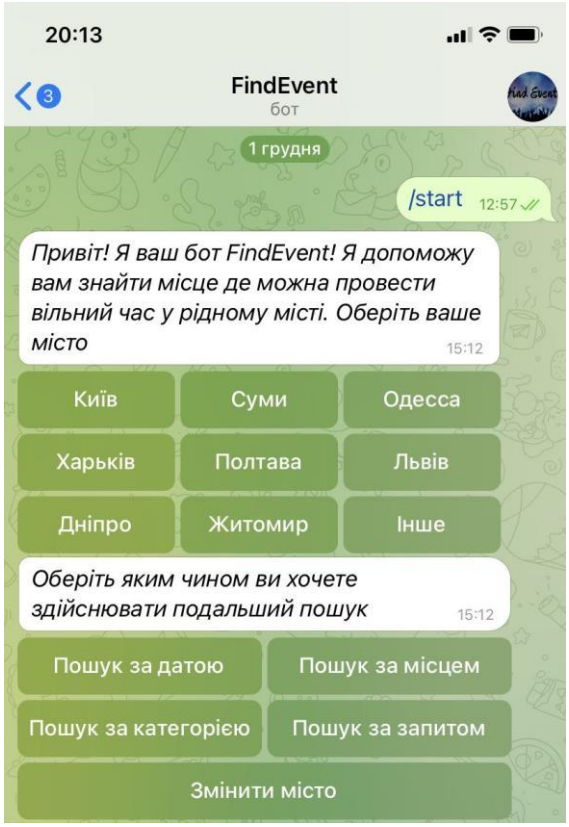

Рис. 3.11. Запуск бота

Крім цього користувачудоступне основне менюбота, яке складається з таких

1. Пошук заходу;
2. Додати захід;
3. Змінити мову;
4. Обране;
5. Рекомендації;
6. Мої нагадування.

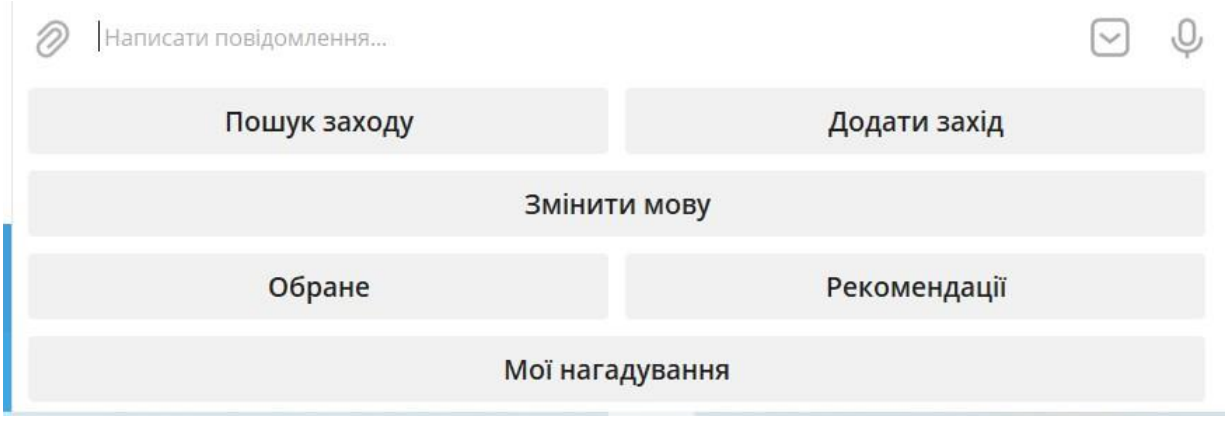

Рис. 3.12. Головне меню бота

Список міст зображено на рисунку 3.13.

Рис. 3.13. Список міст

Далі користувач обирає місто для пошуку або вводить власне місто, вибравши пункт «Інше». Після цього користувачу відображується список критеріїв для подальшого пошуку. До критеріїв пошуку належать:

Пошук за датою;

- Пошук за категорією;
- Пошук за місцем;
- Пошук за запитом.

Список критеріїв зображено на рисунку 3.14.

Рис. 3.14. Список критеріїв для здійснення пошуку

В залежності від вибору критерію, користувачу надається можливість вибору категорії заходу або введення дати, місця, запиту пошуку. Приклад пошуку заходу за датою зображено на рисунку 3.15.

Рис. 3.15. Пошук заходів за датою

Крім цього користувачі бота мають змогу додавання власних заходів. Для цього необхідно вибрати пункт «Додати захід» в головному меню. Додавання заходу користувачем здійснюється в декілька етапів:

1) введення назви заходу;
2) введення опису заходу;
3) введення категорії заходу;
4) введення дати проведення заходу;
5) введення міста проведення заходу;
6) введення посилання на захід;
7) додавання посилання на фото.

Також користувачі мають змогу зміни мови інтерфейсу користувача, обравши пункт «Змінити мову» в головному меню.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи магістра, було реалізовано чат-бот. За мету роботи було визначено розробку чат-бота для месенджеру Telegram, який допоможе користувачам у пошуку найближчих заходів у містах України.

Для досягнення поставленої мети було визначено такі задачі: аналіз предметної області та вимог, порівняння вже існуючих рішень-аналогів, вибір технологій і середовища розробки, розробка чат-боту, розробка супровідної документації.

Також у ході дослідження було обгрунтовано підставу для розробки програмного рішення, було здійснено деталізацію мети та задач проекту, визначено функціональні можливості чат-бота. Для розробки чат-бота було обрано такі технологій та засоби як Python, в якості мови програмування, Sublime Text як середовище розробки.

На етапі проектування було побудовано діаграми, які показують функціональні можливості чат-боту, а саме було побудовано діаграми процесу забезпечення пошуку заходів по містах України у нотаціях IDEF0, діаграму варіантів використання чат бота та модель бази даних.

На останньому етапі було визначено архітектуру чат-бота, здійснено реалізацію чат-бота, а також було продемонстровано основний функціонал бота.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram-бот MoeMisto.ua - твій помічник для вибору дозвілля у Києві! [Електронний ресурс]. – Точка доступу: URL: https://moemisto.ua/kiev/blog/telegram-bot-moemistoua---tviy-pomichnik-dlya-viboru-dozvillya-u-kievi-295.html
2. H. Tian, T. Wang, Y. Liu, X. Qiao, and Y. Li, “Computer vision technology in agricultural automation —A review,” *Information Processing in Agriculture*. 2020, doi: 10.1016/j.inpa.2019.09.006.
3. “Gatwick Airport drone attack: Police have ‘no lines of inquiry’ - BBC News.”. URL: https://www.bbc.com/news/uk-england-sussex-49846450.
4. “Gang who flew drones carrying drugs into prisons jailed - BBC News.” URL: https://www.bbc.com/news/uk-england-45980560. (Дата звернення 17.12.2020).
5. “Drone law in the UK | Nottinghamshire Police.” URL: https://www.nottinghamshire.police.uk/advice/drone-law-uk. (Дата звернення 17.11.2022).
6. “Flying your drone safely and legally.” URL: https://tc.canada.ca/en/aviation/drone-safety/flying-your-drone-safely-legally. (Дата звернення 17.11.2022).
7. P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001, vol. 1, doi: 10.1109/cvpr.2001.990517.
8. N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. I, pp. 886–893, doi: 10.1109/CVPR.2005.177.
9. P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008, doi: 10.1109/CVPR.2008.4587597

10. Y. LeCun *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.
11. W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.
12. C. L. Zitnick and P. Dollár, “Edge boxes: Locating object proposals from edges,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, doi: 10.1007/978-3-319-10602-1_26.
13. C.-J. L. Chih-Wei Hsu, Chih-Chung Chang, “A Practical Guide to Support Vector Classification,” *BJU Int.*, 2008.
14. S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv*. 2018.
15. T. W. Hui, X. Tang, and C. C. Loy, “A lightweight optical flow CNN – Revisiting data fidelity and regularization,” *arXiv*. 2019, doi: 10.1109/tpami.2020.2976928.
16. X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Journal of Machine Learning Research*, 2011.
17. R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014, doi: 10.1109/CVPR.2014.81.
18. “GitHub: The top 10 programming languages for machine learning - TechRepublic.”. URL: https://www.techrepublic.com/article/github-the-top-10-programming-languages-for-machine-learning/.(Дата звернення 09.04.2022).

Додаток А

Лістинг програми

КОД ПРОГРАМИ

Додаток А. Лістинг програмного коду

```
// Tasks

import React, { useEffect, useState } from 'react'; import
'./Tasks.scss';
import { SearchOutlined } from '@ant-design/icons'; import {
Table, Tag, Input, Button, Space } from 'antd'; import Highlighter
from 'react-highlight-words';
import firebase from 'firebase';
import { TasksHeader } from './TasksHeader';
import { TaskDrawerContextState } from './TaskDrawer/TaskDrawerContext'; import {
TaskDrawer } from './TaskDrawer/TaskDrawer';
import { TaskLayout } from './TaskCreate';
import { useFirestoreConnect } from 'react-redux-firebase'; import {
useDispatch, useSelector } from 'react-redux'; import { ItaskStore }
from 'interfaces/TaskInterface';  import { taskStatus } from
'enum/task.enums';
import { SessionsState } from 'interfaces/sessions-state.interface'; import {
deleteTask, setTask, taskDescriptionVisible } from
'./TaskCreate/taskReducer/taskReducer';
import { ToastContainer } from 'react-toastify'; import {
TaskDescription } from
'./TaskDrawer/TaskDescription/TaskDescription';

interface Tasks {  key: string |
 number; taskName: string;
 status: string; updateTime:
 string; author: string;
 maxScore: number;
}

const transformTasks = (tasks: any, docId: string) => { const { id,
  state, lastUpdate, author, maxScore } = tasks;

  key: docId, taskName: id,
  status: state,
  updateTime: lastUpdate, author,
  maxScore };
};
```

```
export const Tasks = () => { const dispatch
  = useDispatch();
  const isLoadingData: boolean = useSelector((state: SessionsState) =>
state.firestore.status.requesting.tasks);

  const [ tasks, setTasks ] = useState<Array<Tasks> | Array<object>>([ {key:
'797984684'} ]);
  const [ selectedRowKeys, setSelectedRowKeys ] = useState<(string | number)[] |
undefined>([]);


  useFirestoreConnect([ { collection: 'tasks' } ]);

 const allTask = useSelector((taskStore: ItaskStore) =>
taskStore.firestore.data.tasks);

  useEffect( () => {
    selectedRowKeys?.length ? dispatch(setTask(allTask[selectedRowKeys[0]])) :
dispatch(deleteTask())
  },
  [allTask, dispatch, selectedRowKeys] );

 useEffect(() => {
  const db = firebase.firestore();


  db.collection('tasks').get().then((query) => {
   query.forEach((doc) => {
    tasks = [ ...tasks, transformTasks(doc.data(), doc.id) ]; });
   setTasks(tasks); });
 }, [allTask]);


 const onSelectChange = (selectedRowKeys: (string | number)[] | undefined) => {
    setSelectedRowKeys(selectedRowKeys);
 };

 const rowSelection = {
  selectedRowKeys, onChange:
```

```
 onSelectChange
};

const [ search, setSearch ] = useState({
 searchText: '',
 searchedColumn: '' });
const { searchText, searchedColumn } = search;

const handleSearch = (selectedKeys: string, confirm: any, dataIndex: string) => {
 confirm();
 setSearch({
  searchText: selectedKeys[0],
  searchedColumn: dataIndex
 }); };

const handleReset = (clearFilters: any) => {
 clearFilters();
 setSearch((prevState: { searchText: string; searchedColumn: string }) => { return {
   ...prevState, searchText:
   ''

 }); };

let searchInput: any;

const getColumnSearchProps = (dataIndex: any) => ({
 filterDropdown: ({ setSelectedKeys, selectedKeys, confirm, clearFilters }: any) => (
  <div style={{ padding: 8 }}> <Input
    ref={(node) => { searchInput
     = node;
    }}
    placeholder={`Search task name`}
    value={selectedKeys[0]}
    onChange={(e) => setSelectedKeys(e.target.value ? [ e.target.value ] : [])}
    onPressEnter={() => handleSearch(selectedKeys, confirm, dataIndex)}
        style={{ width: 188, marginBottom: 8, display: 'block' }}
        key={`SearchInput${dataIndex}`}
   /> <Space>
    <Button type='primary'
     onClick={() => handleSearch(selectedKeys, confirm, dataIndex)}
     icon={<SearchOutlined />}
```

```
      size='small'
           style={{ width: 90 }}
           key='SearchInputButtonSearch'
      > Search
      </Button>
      <Button key='SearchInputButtonReset' onClick={() =>
 handleReset(clearFilters)} size='small' style={{ width: 90 }}>
       Reset </Button>
     </Space> </div>
: undefined }} />,
  onFilter: (value: any, record: any) =>
   record[dataIndex] ?
 record[dataIndex].toString().toLowerCase().includes(value.toLowerCase()) : '',
  onFilterDropdownVisibleChange: (visible: any) => {
   if (visible) {
    setTimeout(() => searchInput.select(), 100); }
  },
  render: (text: any) => searchedColumn ===
   dataIndex ? (
    <Highlighter
     highlightStyle={{ backgroundColor: '#ffc069', padding: 0 }}
     searchWords={[ searchText ]}
     autoEscape
     textToHighlight={text ? text.toString() : ''} />
   ) : ( text
   ) });


  const renderStatus = (status: string) => { let color;

     switch (status) {
       case taskStatus.PUBLISHED: color =
         'green';
         break;

       case taskStatus.PUBLISHED.toUpperCase(): color =
         'green';
         break;

case taskStatus.DRAFT: color = 'orange';
```

```
        case taskStatus.DRAFT.toUpperCase(): color =
          'orange';
              break; default:
          color = 'default'; break;
    }

     return <Tag color={color}>{status?.toUpperCase()}</Tag>; };

    const filtersStatus = [ {
      text: 'Published', value:
        'Published', key: 'Published'
     }, {
      text: 'Draft', value: 'Draft',
          key: 'Draft' },
     {
      text: 'Closed', value:
        'Closed', key: 'Closed'
     } ];

     interface rows {  [key: string]:
       string
     }

    const columns = [ {
title: 'Task Name', dataIndex: 'taskName', key: 'taskName',

     }, {
      title: 'Status', dataIndex:
      'status', key: 'status',
      render: (status: string) => renderStatus(status), filters:
      filtersStatus,
      onFilter: (value: any, record: any) => record.status.indexOf(value) === 0 },
     {
      title: 'Last Update', dataIndex:
      'updateTime', key: 'updateTime'
     }, {
      title: 'Author', dataIndex:
      'author', key: 'author'
     }, {
      title: 'Max Score', dataIndex:
      'maxScore', key: 'maxScore'
```

```
    }, {
    key: 'action',
    render: (values: rows) => ( <Space
      size="middle">
        <Button type='default' onClick={() => dispatch(taskDescriptionVisible(true,
  values.key))} >Description</Button>
      </Space> ),
  }, ];

      autoClose={5000}
      hidePrograssBar={false}
      newestOnTop={false} closeOnClick
      rtl={false} pauseOnFocusLoss
      draggable pauseOnHover
    /> <TaskDrawer>
    <TaskLayout />
   </TaskDrawer>
      <TaskDescription /> <div
   className='tasks'>
    <TasksHeader />
    <div className='tasks-table'> <Table
          loading={isLoadingData}
          dataSource={tasks} columns={columns}
          rowSelection={rowSelection}
        /> </div>
   </div> </TaskDrawerContextState>
            ); };

  // Sessions

  import React, { ReactText } from 'react'; import { Avatar,
  Table, Tag } from 'antd'; import { ColumnsType } from
  'antd/es/table';
  import { useFirestoreConnect } from 'react-redux-firebase'; import {
  useDispatch, useSelector } from 'react-redux'; import styles from
  './Sessions.module.scss';
import { Session } from '../../../../interfaces/app-session.interface'; import { COLORS,
FRIENDLY_STATUS, SessionStatus } from

  session.interface';
  import SessionToolbar from './SessionToolbar/SessionToolbar'; import {
  UserOutlined } from '@ant-design/icons/lib';
  import { setRowSelection } from './SessionsReducer';
```

```
import { SessionHost } from '../../../../interfaces/session-host.interface'; import
SessionForm from './SessionForm/SessionForm';
import { SessionsRecord, SessionsState } from '../../../../interfaces/sessions-
state.interface';

const columns: ColumnsType<Session> = [ {
   key: 'sessionName', title:
   'Session',
   dataIndex: 'sessionName', ellipsis:
   true,
   sorter: (a: Session, b: Session) => {
     const x = a.sessionName.toUpperCase(); const y =
     b.sessionName.toUpperCase(); return x < y ? -1 : x
     > y ? 1 : 0;
   } }, {
   key: 'taskName', title:
   'Task',
   dataIndex: 'taskName', ellipsis:
   true,
   sorter: (a: Session, b: Session) => { const x =
     a.taskName.toUpperCase(); const y =
     b.taskName.toUpperCase(); return x < y ? -1 : x
     > y ? 1 : 0;
   } }, {
   key: 'status', title:
   'Status',
   dataIndex: 'status',
   filters: Object.values(FRIENDLY_STATUS).map(el => ({ text: el, value: el })),

   render: (tag: SessionStatus) => ( <>{tag
     &&
     <Tag color={COLORS[tag]} key={tag}>
       {tag.toUpperCase()}
     </Tag> }
     </> )
  }, {
   key: 'qty',
   title: '# of attendees', dataIndex:
   'qty',
   sorter: (a: Session, b: Session) => a.qty - b.qty, render:
   (tag: number) => (
```

```
      <Tag color='default' key={tag}> {tag}
      </Tag> )
    }, {
      key: 'user', title: 'Lead',
      dataIndex: 'user',
      sorter: (a: Session, b: Session) => {
        const x = a.user?.displayName || 'Anonymous'; const y =
        b.user?.displayName || 'Anonymous'; return x < y ? -1 : x
        > y ? 1 : 0;
      },
      render: (user: SessionHost) => ( <div
        className={styles.user}>
          {user?.photoURL ? <Avatar src={user.photoURL}
  className={styles.user__avatar}/> :
            <Avatar icon={<UserOutlined/>} className={styles.user__avatar}/>}
          {user?.displayName ? <span>{user?.displayName}</span> :
  <span>Anonymous</span>} </div>
)

  ];

  export default function Sessions() { const
    dispatch = useDispatch();
    const sessions: SessionsRecord = useSelector((state: SessionsState) =>
  state.firestore.data.sessions);
    const isLoadingData: boolean = useSelector((state: SessionsState) =>
  state.firestore.status.requesting.sessions);

    useFirestoreConnect([ {
        collection: 'tasks', where: [
          ['state', '==', 'PUBLISHED'] ],
        storeAs: 'publishedTasks' }, {
        collection: 'sessions' }
    ]);

    function getModifiedSessionData(): Session[] { const
      modifiedData: Session[] = [];
      if (sessions) {  Object.keys(sessions).forEach((el: string)
        => {
          if (sessions[el]) {
            const values: FirestoreSessionData = sessions[el];
```

```
        modifiedData.push({
          key: el,
          sessionName: values?.name, taskName:
          values?.task?.taskName, qty:
          values?.attendees?.length || 0,
          status: FRIENDLY_STATUS[values?.status as SessionStatus], user:
          values?.host,
          task: values.task });
                } });
    return modifiedData; }

      return ( <div>
     <div className={styles.toolbar}>
      <SessionToolbar/>
     </div>
     <div className={styles.main}>
      <Table loading={isLoadingData} columns={columns} style={{ width: '100%' }}
           dataSource={getModifiedSessionData()}
           showSorterTooltip={false} pagination={{ pageSize:
           10 }} rowSelection={{
            onChange: (selectedRowKeys: ReactText[]) => {
             dispatch(setRowSelection(selectedRowKeys));
                    } }}
      /> </div>
     <SessionForm/> </div>
  ); }

// Reviews

import React, { useEffect, useState } from 'react'; import styles
from '../Sessions/Sessions.module.scss'; import { Table } from
'antd';
import { AppReviewInterface } from '../../../../interfaces/app-review.interface'; import
firebase from 'firebase';
import { columnsRequests } from './reviewTableDefinition';   import
ReviewsToolBar from './ReviewsToolBar/ReviewsToolBar';
import { ReviewStatusEnum } from '../../../../enum/review-status.enum';
  const [reviews, setReviews] = useState<AppReviewInterface[]>([]); const db =
  firebase.firestore();
  let reviewCounter = 0;

  useEffect(() => {
```

```
    const db = firebase.firestore();
    db.collection('reviews').get()
      .then((reviews) => { let rvs:
        any[] = [];
        reviews.forEach((r) => {
          rvs.push(Object.assign({}, r.data(), {key: r.data().id + r.data().requestId})); })
        setReviews(rvs); })

  }, [reviewCounter])

  const addRowHandler = () => { const
    newRow = {
      key: "0007", id: "0007",
    requestId: "ddddd", author:
    "Mr.Bean",
    state: ReviewStatusEnum.REJECTED }

    db.collection('reviews').add(newRow) .then(() =>
      {
        reviewCounter++; });
  }

        return ( <div>
      <h1>Reviews</h1>
      <ReviewsToolBar
addRow={addRowHandler}

      <div className={styles.main}>
        <Table columns={columnsRequests} style={{ width: '100%' }}
            dataSource={reviews}
            pagination={{ pageSize: 10 }} />
      </div> </div>
  ); };

export default Reviews;

// SelfCheck

import React from 'react';
import { Drawer, Form, Collapse } from 'antd'; import
'antd/dist/antd.css';
import './Selfcheck.scss';
```

```
import FormHeader from '../FormHeader/FormHeader'; import {
useSelector } from 'react-redux';
import { useFirestoreConnect} from 'react-redux-firebase'; import
CategoryItem, { TaskItem } from './CategoryItem';

const { Panel } = Collapse;

interface SelfcheckProps { isVisible:
  boolean,
  hide: () => void,
  setselfGradeValues: (values: any) => void, form: any,
  taskId: string, totalPoints:
  number,
  checkedRequirements: number, setTotalPoints:
  (number: number) => void,
  setCheckedRequirements: (number: number) => void, }

interface TasksState { firestore: {

    } });
   setTotalPoints(totalPoints);
   setCheckedRequirements(checkedRequirements);
  }

  const addSelfGrade = (values: any) => {
    Object.keys(values).forEach((key: string) => {
      if (values[key] === undefined) { delete
        values[key];
      } });

    setselfGradeValues({ ...values,
      totalPoints,
      checkedRequirements });
  }

  return ( <Drawer
      mask={false} closable={false}
      visible={isVisible}
      placement='left' width={600}
      title={
        <FormHeader title="Self-check" onClose={handleClose} form={form}/> }
```

```
    >
    <div className="self-check">
      <Form name="self-check" form={form} onFinish={onFinish}
onValuesChange={onValuesChange} initialValues={undefined} >
        <div className="self-check__current-values"> <h3>Total
          points: {totalPoints}/{isVisible &&
tasks[taskId].maxScore}</h3>

        </div>
        <p>{(tasks && isVisible) && tasks[taskId].description}</p> <Collapse
        bordered={false} style={{backgroundColor: 'white'}}>
         {(tasks && isVisible) &&
            tasks[taskId].categoriesOrder.map((category: string) => (
              <Panel header={category} key={category}>
               {tasks[taskId].items.map((item: TaskItem, ind: number) => {
                 return item.category === category && <CategoryItem item={item}
key={item.id} />;
               } )}
              </Panel> ))
         } </Collapse>
      </Form> </div>
    </Drawer>
  ); }

export default Selfcheck;

// Requests

import React, { useState } from 'react'; import styles
from './Requests.module.scss'; import './Requests';

import HeaderRequests from './HeaderRequests/HeaderRequests';   import
TopPanelRequests from './TopPanelRequests/TopPanelRequests'; import
TableRequests from './TableRequests/TableRequests';

import { Button, Form } from 'antd';
import { EditOutlined } from '@ant-design/icons';
import RequestForm from '../RequestForm/RequestForm';

        </div>
        <p>{(tasks && isVisible) && tasks[taskId].description}</p> <Collapse
        bordered={false} style={{backgroundColor: 'white'}}>
```

```
        {(tasks && isVisible) &&
            tasks[taskId].categoriesOrder.map((category: string) => (
              <Panel header={category} key={category}>
                {tasks[taskId].items.map((item: TaskItem, ind: number) => {
                  return item.category === category && <CategoryItem item={item}
key={item.id} />;
                } )}
              </Panel> ))
        } </Collapse>
      </Form> </div>
    </Drawer>
  ); }
```

Додаток Б

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Кваліфікаційна робота.doc	Пояснювальна записка. Документ Word.
Кваліфікаційна робота.pdf	Пояснювальна в форматі PDF
Программа	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація.ppt	Презентація