

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОШУКУ ДРУЗІВ

(назва теми)

У МІСТАХ ПРИ ПЕРЕЇЗДІ. SOFTWARE FOR MAKING

FRIENDS IN CITIES WHEN MOVING

Виконав(ла): студент(ка) 4 курсу, групи КНТ-110

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПОГРІБНЯК В.М.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ КНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 121 Інженерія програмного забезпечення _____
(код і найменування)
Освітня програма (спеціалізація) _____ Інженерія програмного забезпечення _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф. _____
Сергій СУББОТІН
« _____ » _____ 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПОГРІБНЯКА Владислава Миколайовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для пошуку друзів у містах при переїзді. Software for Making Friends in Cities When Moving

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович _____,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 24 » квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Владислав ПОГРІБНЯК
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 88 с., 21 рис., 2 табл., 3 дод., 7 джерел.

МІСТО, ПОШУК ДРУЗІВ, ПЕРЕЇЗД, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РОБОТА, ОСВІТА, БАЗА ДАНИХ.

Об'єкт роботи – процес пошуку друзів у містах при переїзді. Предмет роботи – програмне забезпечення для пошуку друзів у містах при переїзді. Мета роботи – розробити програмне забезпечення для пошуку друзів у містах при переїзді, щоб забезпечити підтримку інформування людей під час даного процесу.

Матеріали, методи та технічні засоби: мова Python, система MySQL, фреймворк Django.

Результати. Основним результатом є програма, яка забезпечує пошук друзів у містах, в які користувачі бажають переїхати, та надає можливість підтримки цього процесу щодо публікації тих запитань, які виникають у користувачів та надання на них відповідей. Реалізовано можливості прив'язування запитань щодо переїзду в нове місто до професій, до сфер роботи, до параметрів навчання, до хобі. Створено потік запитань щодо переїзду в місто, де проживає авторизований користувач і щодо якого він може розглядатися як експерт, з можливістю відкидання запитань або створення відповідей на них.

Висновки. Розробку програми виконано через забезпечення етапів, що охоплювали визначення особливостей аналогів і порівняння з розробкою, виділення основних засобів розробки, реалізацію моделей, представлень, шаблонів для забезпечення всіх функціональних можливостей.

Галузь використання. Персональне використання, забезпечення комунікації людей між різними містами щодо питань, пов'язаних з переїздом.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 88 pages, 21 figures, 2 tables, 3 appendixes, 7 sources.

CITY, MAKING FRIENDS, MOVING, SOFTWARE, WORK, EDUCATION, DATABASE.

The object of the work is the process of making friends in cities when moving. The subject of the work is software for making friends in cities when moving. The goal of the work is to develop software for making friends in cities when moving to provide support for informing people on this process.

Materials, methods and technical means: Python language, MySQL system, Django framework.

The results. The main result is an application that provides the search for friends in the cities where users want to move, and provides the possibility to support this process by publishing the questions that arise from users and providing answers to them. It is possible to link questions about moving to a new city to professions, fields of work, study options, and hobbies. A stream of questions about moving to a city where an authorized user lives and can be considered an expert, was created with the ability to dismiss questions or create answers to them.

Conclusions. The development of the program was carried out through the provision of stages, which covered the definition of the features of analogues and comparison with the development, the selection of the main means of development, the implementation of models, views, templates to ensure all functional capabilities.

Field of use. Personal use, enabling people to communicate between different cities on matters related to moving.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Визначення концепції пошуку друзів у містах при переїзді	9
1.2 Порівняння засобів аналогів щодо пошуку друзів у містах при переїзді.....	11
1.3 Висновки за розділом 1	12
2 Розробка архітектури програми.....	13
2.1 Вибір засобів проєктування	13
2.2 Сценарії використання програми	14
2.3 Структура бази даних	17
2.4 Висновки за розділом 2	32
3 Реалізація програми	33
3.1 Порядок функціонування програми.....	33
3.2 Результати реалізації програмних одиниць додатку	34
3.3 Висновки за розділом 3	48
4 Експлуатація та тестування програми	49
4.1 Призначення програми	49
4.2 Умови виконання програми	49
4.3 Виконання і тестування програми.....	50
4.4 Обробка збійних ситуацій	54
4.5 Висновки за розділом 4	55
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	58
Додаток Б Текст програми	63
Додаток В Слайди презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;
ВФ – вебфреймворк.

ВСТУП

Багато українців вимушено відчуло на собі проблеми, пов'язані з переїздом в інші міста. Знайти нове коло спілкування може бути додатковою проблемою, однак отримати початкову інформацію, зрозуміти, наскільки дане місто взагалі підходить для способу життя людини і чи не обрати інше, є дуже важливими запитаннями. Для відповідей на такі запитання є дуже корисне джерело – люди, які в цьому місті проживають.

Однак запитання можуть бути спеціалізованими: стосуватися навчання, роботи за певними професіями або роботи загалом, певних хобі (як наприклад, готовності міського простору для занять бігом на вулицях міста). У таких випадках важливо віднайти цільову аудиторію таких запитань, які здатні дати потрібні відповіді. Така орієнтованість дозволяє створити певні спільноти в програмі, звертаючись до відповідного адресата та запрошуючи його дати відповідь, а за рахунок привертання учасників таких спільнот (наприклад, ІТ-спільноти) отримувати корисні відповіді, а в подальшому можливо встановлювати більш тісний контакт, знаходити друзів.

Усе це сукупно повинно дозволити забезпечити цілісну підтримку людей під час переїзду в нове місто, в тому числі під час формування таких планів на переїзд. Звичайно, що в сучасному світі це потребує створення програмного інструменту для пошуку друзів у містах при переїзді.

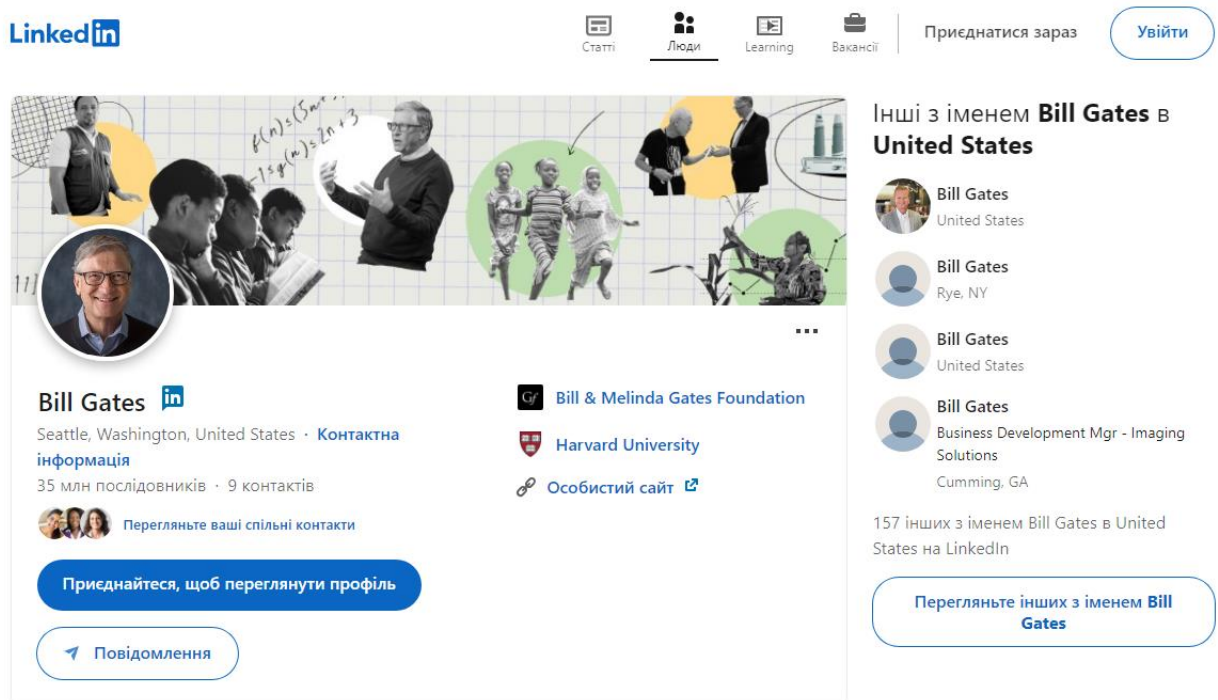
Тож мета роботи – розробити програмне забезпечення для пошуку друзів у містах при переїзді, щоб забезпечити підтримку інформування людей під час даного процесу

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення концепції пошуку друзів у містах при переїзді

Необхідно спершу зазначити, що попередній огляд існуючих програмних засобів продемонстрував, що готового рішення, яке б повністю розв'язувало задачу пошуку друзів у містах при переїзді зокрема стосовно інформування користувачів на основі системи запитання-відповідь, на даний момент не існує. Проте звичайно є різні вебдодатки, які частково цю проблему вирішують.

LinkedIn [1] (рис. 1.1) дозволяє встановлювати ділові контакти, відповідно це може сприяти розв'язуванню тих проблемних питань, які є у людини. Встановивши контакт з людиною з іншого міста, може уточнити професійні питання, які стосуються переїзду, особливостей ринку в певному місті.



Загальна інформація

Co-chair of the Bill & Melinda Gates Foundation. Founder of Breakthrough Energy... дивитися більше

Рисунок 1.1 – Вебдодаток LinkedIn

Проте все це реалізується у формі подальших індивідуальних контактів, єдиного простору для того, щоб сформулювати запитання, а всі бажаючі люди, які мають відповідну інформацію, могли б на ці запитання відповісти, немає.

Якщо проблема стосується пошуку друзів, то найвідомішою соціальною мережею для цього є Facebook [2] (рис. 1.2). Щодо Facebook важливо враховувати універсальність, величезну кількість користувачів, але при тому розуміти і обмеженість його стосовно саме питань переїзду в інше місто.

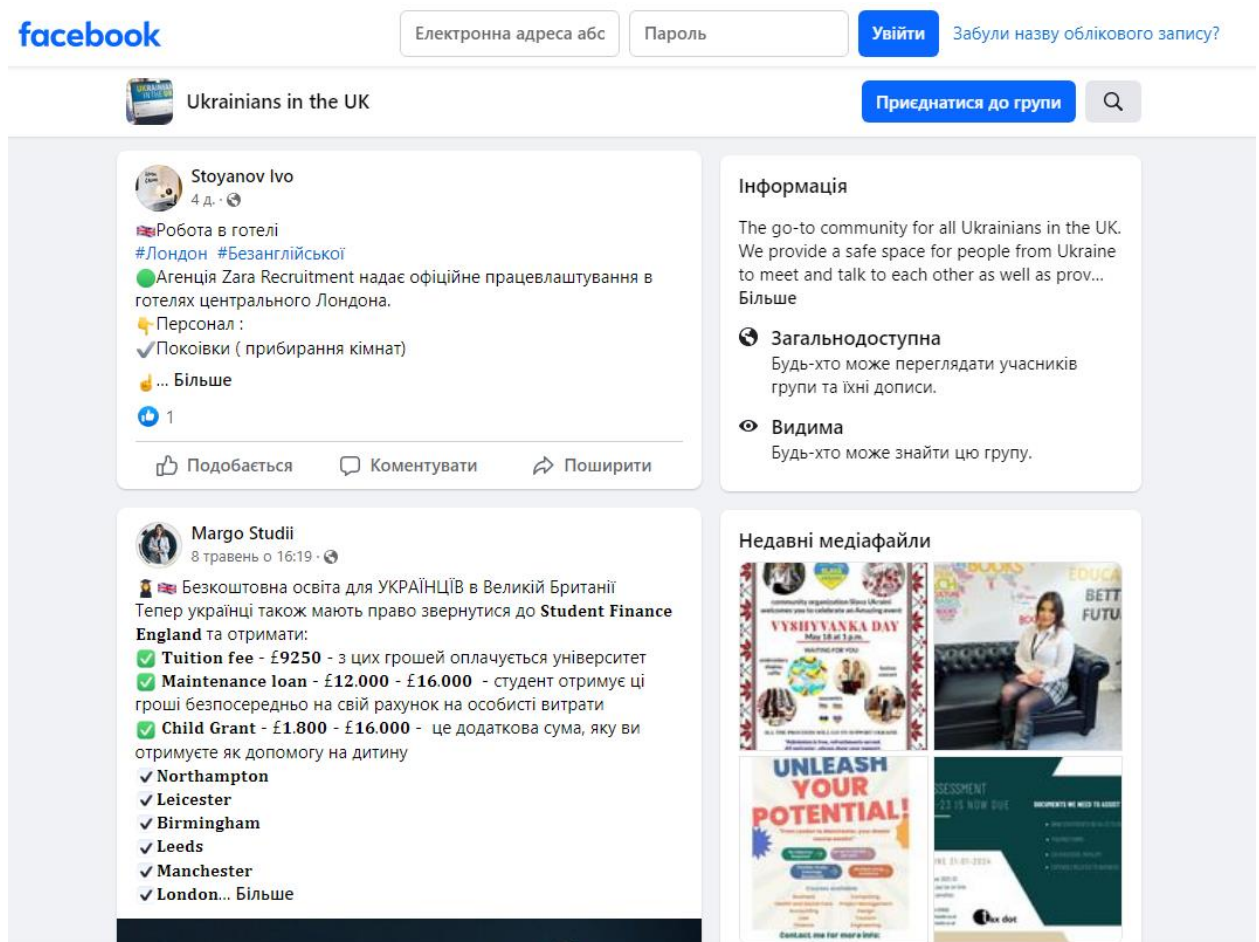


Рисунок 1.2 – Вебдодаток Facebook

Фактично Facebook має велику кількість груп, але серед груп є і такі, які наприклад, стосуються проживання людей певної національності в певній країні, місті. Тому до такої групи можна доєднатися, поставити запитання, однак проблемою є те, що в активній групі кількість запитань може бути

великою, шукати за певними тематики запитання неможливо (є тільки загальний пошук), до того ж всі запитання змішені з будь-якою інформацією, яка надається, тобто є єдина стрічка, яка включає різномірну інформацію, а це не сприяє досягненню і отриманню потрібної інформації в підсумку.

1.2 Порівняння засобів аналогів щодо пошуку друзів у містах при переїзді

Порівняння LinkedIn, Facebook та CITYMover, що є назвою програми, яка створюється, було проведено за послідовністю критеріїв (табл. 1.1).

Таблиця 1.1 – Порівняння засобів аналогів щодо пошуку друзів у містах при переїзді

Критерій	LinkedIn	Facebook	CITYMover
Пошук запитань щодо переїзду в інше місто	—	В межах спільноти серед всіх публікацій	+
Наявність окремого потоку запитань щодо переїзду в інше місто	—	—	+
Встановлення контактів, у тому числі в обмежених спільнотах	+	+	+
Деталізоване і структуроване представлення власної освіти, досвіду роботи, хобі	+	—	+
Зв'язування запитань з професією, хобі	—	—	+

CITYMover повинен надати можливість переглядати окремий потік, який формується з запитань виключно за містом, в якому проживає і щодо якого має необхідну інформацію користувач, який авторизувався. Це дозволяє консультувати людей, які збираються переїхати у це місто, а тим людям відповідно отримувати необхідну інформацію. Така можливість відсутня в інших додатках і може розглядатися тільки через пошук в межах спільноти серед всіх публікацій у Facebook, однак такий пошук не виділяє запитання окремо. LinkedIn має важливу особливість стосовно деталізованого і структурованого представлення власної освіти, досвіду роботи, що дозволяє сприймати кожного користувача як учасника певної обмеженої спільноти (працівників), відповідно це створює певний важливий акцент, який за рахунок зв'язування такої інформації з запитаннями має дозволити досягати більшої ефективності отриманих відповідей на запитання у додатку, що розробляється.

1.3 Висновки за розділом 1

Виконаний огляд програмних інструментів, придатних для пошуку друзів у містах при переїзді, встановив, що такі існуючі інструменти, як LinkedIn, Facebook мають певні окремі можливості, які сприяють пошуку друзів або інформації при переїзді в інше місто, проте не є цілеспрямованими для досягнення цього. Тому виділення корисних особливостей з цих програм та об'єднання їх з концепцією запитань є важливим для розв'язання задачі пошуку друзів у містах при переїзді.

Завдання, які мають бути розв'язані в роботі:

- проаналізувати програми для пошуку друзів у містах при переїзді;
- визначити засоби проєктування програми;
- виділити вимоги до програми пошуку друзів у містах при переїзді;
- виконати реалізацію програми пошуку друзів у містах при переїзді.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

Для розробки вебдодатку на даний момент важливим є вибір вебфреймворку (ВФ), а зважаючи на популярність рішень на PHP та загальну популярність Python порівняння було зведено до лідерів відповідних груп – Laravel [3]-[4] і Django [5]-[6] (табл. 2.1).

Таблиця 2.1 – Порівняння ВФ для розробки програми для пошуку друзів у містах при переїзді

Критерій	Laravel	Django
Мова програмування	PHP	Python, що має більшу простоту синтаксису, легкість вивчення тощо
Продуктивність	Нижча за рахунок великої кількості вбудованих засобів, що можуть бути непотрібними	Вища
Підтримувані бази даних	MySQL, PostgreSQL, SQLite	MySQL, Oracle, MariaDB, PostgreSQL, SQLite
Архітектура	Model-View-Controller	Model-View-Template
Рівень безпеки	Підвищується, але потребує пошуку та контролю	Високий за рахунок наявних інтегрованих засобів, що охоплюють величезну кількість проблем
Підтримка	Добра	Дещо краща за рахунок широкого кола документації

Обрано було Django, врахувавши те, що це ВФ з більшою продуктивністю, кращими характеристиками основної мови, вищим базовим рівнем безпеки та дещо кращою підтримкою.

2.2 Сценарії використання програми

Користувачі не розділялись на різні групи чи ролі в програмі, адже потенційно кожен з них може і надавати інформацію, і потребувати інформацію про життя в містах. Кожен користувач має свій унікальний досвід та може надати інформацію про місто, адже десь він проживає у поточний момент. Відповідно інші користувачі можуть задумуватися щодо переїзду саме в те місто, де цей користувач живе. Тому встановлення контакту є важливим. Але в різні моменти кожен користувач може виступати з будь-якої сторони взаємодії. І логічним є не змушувати користувачів для цього створювати різні облікові записи. Сценарії користувача:

- формування потоку користувача з запитань щодо переїзду;
- створення відповіді на запитання щодо переїзду;
- пошук запитань користувачів щодо переїзду;
- відображення списку опублікованих користувачем запитань щодо переїзду;
- публікація запитання щодо переїзду;
- відображення картки користувача;
- визначення даних картки користувача, включаючи визначення досвіду роботи користувача, визначення його хобі, результатів навчання, мов для комунікації;
- авторизація;
- відображення публікації запитання щодо переїзду в нове місто;
- відкладання картки потенційного друга;
- відображення відкладених карток потенційних друзів;
- подання запиту на дружбу в додатку з користувачем;

- прийняття запиту на дружбу в додатку;
- відображення нових ініційованих та адресованих запитів на дружбу;
- відображення друзів користувача в додатку;
- відображення відправлених повідомлень між друзями, включаючи відправлення на пряму повідомлення другу;
- відображення списку опублікованих користувачем відповідей на запитання щодо переїзду;
- відзначення корисною відповіді на запитання щодо переїзду в нове місто;
- відображення списку корисних для користувача відповідей (рис. 2.1).

Альтернативою до сценарію відзначення корисною відповіді на запитання щодо переїзду в нове місто є скасування відзначення корисною відповіді на запитання щодо переїзду в нове місто, коли таке відзначення відбулось, але думка користувача змінилась, тобто так чи інакше виявилось, що таке відзначення є помилковим.

Альтернативою до сценарію прийняття запиту на дружбу в додатку є відхилення запиту на дружбу в додатку, що відбувається тоді, коли користувач не бажає продовжувати спілкування на основі повідомлень, які відправляються на пряму один одному.

Альтернативою до сценарію відкладання картки потенційного друга є вилучення відкладеної картки потенційного друга, коли користувач втратив цікавість до інформації від користувача за відкладеною карткою.

Альтернативою до сценарію публікації запитання щодо переїзду є закриття запитання щодо переїзду, коли остаточну відповідь на запитання було отримано і його автор більше не потребує відповідей або актуальність запитання була втрачена.

Альтернативою до сценарію створення відповіді на запитання щодо переїзду є скасування відображення запитання щодо переїзду в нове місто в потоці, коли користувач не бажає давати відповідь з тієї чи іншої причини і при цьому і не бажає, щоб це запитання виникало перед ним знов.

Альтернативою до сценарію авторизації є реєстрація, яка виникає за умов потреби в авторизації, але відсутності облікового запису для роботи з програмою.



Рисунок 2.1 – Діаграма прецедентів програми для пошуку друзів у містах при переїзді

2.3 Структура бази даних

Прямою вимогою до визначення структури бази даних (БД) є склад функціональних вимог до програми. Тому базуючись на результатах виконання підрозділу 2.2, було створено ряд таблиць, які забезпечують реалізацію реляційної БД на основі системи MySQL, яка на даний момент є високопродуктивною безкоштовною системою, яка для роботи з БД надає широкий серед альтернатив ряд засобів.

Міста, містечка, селища, куди переїжджають люди та де шукають собі друзів, визначені таблицею Mappoint з рядом полів:

- id – ідентифікатор міста (містечка або селища), доступного для роботи;
- country – країна, в якій знаходиться це місто (країни вирішено було задати програмно відповідним набором, оскільки цей набір залишається більш-менш статичним і може бути заданий початково в програмі, навіть якщо в цій країні робота програми і не буде підтримуватись, тобто зміна набору відбувається тільки в тому випадку, коли змінюються країни, що може бути виконано і програмно, зважаючи на рідкість);
- toponame – офіційна назва міста (селища або містечка);
- centerlat та centerlong – географічна локація (широта і довгота) центра міста (селища або містечка).

Картки користувачів програми, що можуть бути комусь потенційними або фактичними друзями, проживаючи в цільовому місті, або знаходитись в пошуку друзів або інформації від них у цільовому місті, визначені таблицею Friendapplicant з рядом полів:

- id – ідентифікатор картки користувача, який може бути потенційним другом;
- softname – ім'я користувача, який може бути потенційним другом в програмі, яким він відображається;

– appuser – користувацький запис, за яким цей потенційний друг працює в Django, а відповідно і в програмі;

– marpoint – місто (містечко або селище), де на даний момент проживає користувач, який може бути потенційним другом: якщо користувач планує переїзд, то зазначається саме місто, де він проживає, а не куди збирається переїхати;

– birthdate – дата народження користувача, який може бути потенційним другом;

– gender – стать користувача, який може бути потенційним другом;

– mainphoto – фотографія, на якій показаний користувач, який може бути потенційним другом;

– firstlogin – час події першої авторизації користувача, який може бути потенційним другом, в програмі.

Додаткові фотографії користувачів, які можуть бути потенційними друзями, визначені таблицею Friendapplicantphoto з рядом полів:

– id – ідентифікатор додаткової фотографії користувача, який може бути потенційним другом;

– friendapplicant – користувач, який може бути потенційним другом і який зображений на фотографії;

– friendphoto – сама фотографія, на якій показаний користувач, який може бути потенційним другом;

– addeddate – час події додавання фотографії в програму.

Friendapplicant і Friendapplicantphoto є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один користувач, який може бути потенційним другом, може мати від 0 до певної кількості додаткових фотографій, де він зображений, а одна фотографія – відображення одного користувача, який може бути потенційним другом.

Професії, які можуть задаватися в програмі, визначені таблицею Jobprofession з рядом полів:

– id – ідентифікатор професії;

- mainname – основна назва професії;
- pseudoname – додаткова назва професії (за наявності): наприклад, може бути поширеною і оригінальна назва, і англomовна назва професії, тому краще надати спосіб шукати одночасно за обома варіантами.

Історія досвіду роботи користувача, який може бути потенційним другом, визначена таблицею Jobhistory з рядом полів:

- id – ідентифікатор запису історії досвіду роботи користувача, який може бути потенційним другом;
- friendapplicant – користувач, для якого цей досвід роботи визначений та який може бути потенційним другом;
- jobprofession – професія, за якою працював на цьому підприємстві цей користувач;
- publicenterprisesphere – сфера роботи підприємства-роботодавця, до якої відносилась виконував робота користувачем, який може бути потенційним другом: одна й та сама професія може належати до різних сфер і мати свою специфіку (розробник в ІТ-компанії та розробник на металургійному підприємстві), окрім того одне підприємство може надавати послуги, виконувати роботи в різних сферах (якщо це велика компанія, яка можливо об'єднує в своїй структурі інші компанії);
- experiencestart – дата події влаштування на цю роботу;
- experienceend – дата події звільнення з цієї роботи (якщо дата не задана, то має вважатися, що робота є поточною);
- responsibility – опис особливостей робочих обов'язків, які виконував працівник.

Friendapplicant і Jobhistory є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один користувач, який може бути потенційним другом, може мати від 0 до певної кількості записів історії досвіду роботи, а кожен досвід роботи є досвідом роботи тільки одного користувача, який може бути потенційним другом.

Jobprofession і Jobhistory є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що одна професія може мати від 0 до певної кількості записів історії досвіду роботи різних користувачів, а кожен досвід роботи є досвідом роботи тільки за однією професією.

Publicenterprisesphere і Jobhistory є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що одна сфера підприємства-роботодавця може мати від 0 до певної кількості записів історії досвіду роботи різних користувачів, а кожен досвід роботи є досвідом роботи тільки за однією сферою роботи одного підприємства-роботодавця.

Сфери роботи визначені таблицею Jobsphere з рядом полів:

- id – ідентифікатор сфери роботи;
- spherename – назва сфери роботи.

Підприємства-роботодавці визначені таблицею Publicenterprise з рядом полів:

- id – ідентифікатор підприємства-роботодавця;
- officialname – офіційна назва підприємства-роботодавця;
- nonofficialname – неофіційна назва підприємства-роботодавця (додаткова назва для зручності пошуку);
- officemappoint – місто (селище, містечко), де знаходиться головний офіс підприємства;
- specialtydetails – особливості роботи підприємства-роботодавця, надаваних послуг;
- officialsite – офіційний сайт підприємства-роботодавця.

Mappoint і Publicenterprise є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що одне місто може мати від 0 до певної кількості підприємств-роботодавців, а кожне підприємство-роботодавець може мати головний офіс лише в одному місті.

Сфери, в яких працюють підприємства-роботодавці, визначені таблицею Publicenterprisesphere з рядом полів:

- id – ідентифікатор сфери підприємства-роботодавця;

- publicenterprise – підприємство-роботодавець, що працює в цій сфері;
- jobsphere – сфера, в якій працює підприємство-роботодавець.

Publicenterprisesphere необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Jobsphere і Publicenterprise, що виник через потребу забезпечити роботу підприємств-роботодавців у різних сферах, а різним сферам бути представленими від 0 до певної кількості підприємств-роботодавців.

Хобі, які для себе можуть визначати користувачі, що можуть бути потенційними друзями, визначені таблицею Hobby з рядом полів:

- id – ідентифікатор хобі;
- mainname – основна назва хобі;
- pseudoname – додаткова назва (наприклад, англійською мовою або певний синонім, що має спростити пошук за рахунок рівноцінного використання назв).

Хобі користувачів, які можуть бути потенційними друзями, визначені таблицею Friendapplicanthobby з рядом полів:

- id – ідентифікатор хобі користувача, який може бути потенційним другом;
- friendapplicant – користувач, який може бути потенційним другом та який займається цим хобі;
- hobby – хобі, яким займається користувач, який є потенційним другом;
- experiencestart – час події, коли вперше користувач, який може бути потенційним другом, зайнявся цим хобі;
- details – особливості досвіду заняття цим хобі користувачем, який може бути потенційним другом.

Friendapplicanthobby необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendapplicant і Hobby, що виник через потребу забезпечити визначення користувачем, який може бути потенційним другом,

декількох хобі, які в нього є, а хобі при цьому можуть вибиратися різними такими користувачами додатку.

Заклади освіти, доступні в програмі, визначені таблицею Zvo з рядом полів:

- id – ідентифікатор закладу освіти;
- officialname – офіційна назва закладу освіти;
- zvolevel – рівень закладу освіти;
- nonofficialname – неофіційна назва закладу освіти або колишня назва закладу освіти, якщо заклад змінював назву (наприклад, Запорізький національний технічний університет в якості nonofficialname та Національний університет «Запорізька політехніка» в якості officialname);
- officemappoint – місто, де знаходиться головний офіс закладу освіти;
- details – особливості навчання, організації роботи закладу освіти;
- officialsite – офіційний вебсайт закладу освіти.

Mappoint і Zvo є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що одне місто може мати від 0 до певної кількості закладів освіти, а кожен заклад освіти визначається тільки одним містом розташування.

Результати навчання користувачів, які можуть бути потенційними друзями, визначені таблицею Graduation з рядом полів:

- id – ідентифікатор результату навчання користувача, який може бути потенційним другом;
- friendapplicant – користувач, який може бути потенційним другом і який навчався за цією програмою;
- zvo – заклад освіти, де навчався користувач, який може бути потенційним другом;
- experiencestart – дата події вступу до закладу освіти;
- experienceend – дата події завершення навчання в закладі освіти;
- jobsphere – сфера роботи, за якою навчання відбувалось;
- degreetype – рівень освіти, за яким здійснювалось навчання;

– state – стан навчання (успішно завершено, відбувається, незавершено);

– degree name – назва отриманої кваліфікації.

Friendapplicant і Graduation є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один користувач, який може бути потенційним другом, може мати від 0 до певної кількості результатів навчання, але кожен результат навчання є результатом одного конкретного користувача.

Zvo і Graduation є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один заклад освіти може навчати від 0 до певної кількості студентів, забезпечуючи їм відповідні результати, але кожен результат навчання надається тільки одним закладом освіти.

Зважаючи на різні міста, а потенційно і різні країни, користувачі можуть розмовляти різними мовами, а тому ця характеристика для кожного потенційного друга є важливою, адже дозволяє визначити, чи зможуть люди між собою розмовляти. Доступні для вибору в програмі мови для комунікації визначені таблицею Communicationlanguage з рядом полів:

– id – ідентифікатор мови для комунікації;

– language name – назва мови для комунікації.

Мови, якими володіють користувачі, які можуть бути потенційними друзями, визначені таблицею Friendapplicantlanguage з рядом полів:

– id – ідентифікатор мови, якою володіє користувач, який може бути потенційним другом;

– friendapplicant – користувач, який може бути потенційним другом і який володіє цією мовою;

– communicationlanguage – мова для комунікації, якою володіє цей користувач, який може бути потенційним другом;

– knowledgelevel – рівень знання цієї мови для комунікації користувачем, який може бути потенційним другом.

Friendapplicantlanguage необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendapplicant і Communicationlanguage, що виник через потребу забезпечити володіння користувачем, який може бути потенційним другом, декількома мовами, а не тільки однією, але ж до того ж і мови для комунікації можуть визначатися стосовно різних користувачів.

Відкладання карток потенційних друзів визначені таблицею Friendapplicantmark з рядом полів:

- id – ідентифікатор відкладання картки потенційного друга;
- friendapplicant – картка потенційного друга, яку відклав користувач з наміром переглянути її пізніше;
- friendsaver – картка користувача, який відклав цю картку потенційного друга;
- markdate – час події відкладання картки потенційного друга.

Friendapplicantmark необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendapplicant і Friendapplicant, що виник через потребу забезпечити відкладання однієї картки потенційного друга різними користувачами, а кожен такий користувач, який може бути потенційним другом, може відкладати різні картки.

Публікації запитань користувачів щодо переїзду в нове місто визначені таблицею Friendpost з рядом полів:

- id – ідентифікатор публікації запитання користувача щодо переїзду в нове місто;
- friendapplicant – користувач, який може бути потенційним другом та який поставив певне запитання стосовно переїзду в нове місто;
- posttext – текст запитання стосовно переїзду в нове місто;
- postdate – дата події публікації запитання користувачем щодо переїзду в нове місто;
- mappoint – місто, куди має намір переїхати користувач, який може бути потенційним другом;

– toarchive – позначка, що публікація запитання користувача щодо переїзду в нове місто має бути заархівована.

Friendapplicant і Friendpost є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один користувач може створити від 0 до певної кількості публікацій з запитаннями щодо переїзду, проте авторство завжди належить лише одному користувачу, який може бути потенційним другом.

Mappoint і Friendpost є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що щодо переїзду в одне місто можуть виникати різні запитання і відбуватися публікації цих запитань, а кожне запитання стосується тільки одного міста, куди на даний момент планує переїхати користувач.

Скасування відображення запитання щодо переїзду в нове місто для користувача, який потенційно міг би бути другом, визначені таблицею Friendpostminus з рядом полів:

- id – ідентифікатор скасування відображення запитання щодо переїзду в нове місто для користувача, який потенційно міг би бути другом;
- friendpost – публікація запитання користувача щодо переїзду в нове місто, відображення якої скасував для себе цей користувач;
- friendapplicant – користувач, який може бути потенційним другом, але який скасував відображення для себе запитання щодо переїзду в нове місто іншого користувача.

Friendpostminus необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendapplicant і Friendpost, що виник через потребу забезпечити скасування відображення одного й того самого запитання щодо переїзду в нове місто для різних користувачів, але і одночасну потребу для одного й того самого користувача скасовувати відображення різних запитань щодо переїзду в нове місто.

Фотографії з публікацій запитань щодо переїзду в нове місто визначені таблицею Friendpostphoto з рядом полів:

– id – ідентифікатор фотографії з публікації запитання щодо переїзду в нове місто;

– friendpost – публікація запитання користувача щодо переїзду в нове місто, до якої фотографія віднесена;

– photo – вміст фотографії, що відноситься до цієї публікації запитання користувача щодо переїзду в нове місто.

Friendpost і Friendpostphoto є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що публікація запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості фотографій, а одна фотографія – додаток до тільки однієї публікації запитання користувача щодо переїзду в нове місто.

Хобі, яких стосується запитання щодо переїзду в нове місто та які віднесені до цього запитання його автором, визначені таблицею Friendposthobby з рядом полів:

– id – ідентифікатор хобі, яке відноситься до запитання щодо переїзду в нове місто;

– friendpost – публікація запитання користувача щодо переїзду в нове місто, до якої віднесено це хобі;

– hobby – хобі, яке віднесено до цієї публікації запитання користувача щодо переїзду в нове місто.

Friendposthobby необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendpost і Hobby, що виник через потребу забезпечити віднесення публікації запитання користувача щодо переїзду в нове місто до різних хобі, а кожному хобі забезпечити можливість багаторазового вибору для різних публікацій запитання користувача щодо переїзду в нове місто.

Якщо запитання щодо переїзду відноситься до навчання, то з публікацією може бути співставлено параметри навчання, визначені таблицею Friendpostgraduation з рядом полів:

– id – ідентифікатор параметрів навчання, визначених стосовно публікації запитання щодо переїзду в нове місто;

– friendpost – публікація запитання щодо переїзду в нове місто, до якої віднесено ці параметри навчання;

– jobsphere – сфера роботи, за якою здійснюється навчання;

– zvo – заклад освіти, стосовно якого поставлено запитання щодо переїзду в нове місто (якщо воно відноситься до конкретного закладу, або порожнє в протилежній ситуації).

Friendpost і Friendpostgraduation є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що публікація запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості пов'язаних професій (наприклад, декілька суміжних професій), а кожна професія може представлятися в різних публікаціях запитань користувачів щодо переїзду в нове місто.

Jobsphere і Friendpostgraduation є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що публікація запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості пов'язаних сфер роботи, а кожна сфера може представлятися в різних публікаціях запитань користувачів щодо переїзду в нове місто.

Zvo і Friendpostgraduation є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що публікація запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості пов'язаних закладів освіти, а кожен заклад освіти може представлятися в різних публікаціях запитань користувачів щодо переїзду в нове місто.

Якщо запитання щодо переїзду відноситься до роботи, то в цьому розрізі головною інформацією, яку може надати потенційний друг, є інформація щодо роботи за професією. Те, якої професії стосується запитання, визначено таблицею Friendpostprofession з рядом полів:

– id – ідентифікатор професії, визначеної стосовно публікації запитання щодо переїзду в нове місто;

– friendpost – публікація запитання щодо переїзду в нове місто, до якої віднесено цю професію;

– jobprofession – професія, яку віднесено до цієї публікації запитання щодо переїзду в нове місто.

Friendpostprofession необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Jobprofession і Friendpost, що виник через потребу забезпечити віднесення публікації запитання користувача щодо переїзду в нове місто до різних сфер (але роботи, не навчання), а кожній сфері забезпечити можливість багаторазового вибору для різних публікацій запитання користувача щодо переїзду в нове місто.

Також запитання може стосуватися загалом певної сфери роботи, тоді віднесення публікації запитання щодо переїзду в нове місто до певної сфери роботи визначено таблицею Friendpostjobsphere з рядом полів:

– id – ідентифікатор сфери роботи, визначеної стосовно публікації запитання щодо переїзду в нове місто;

– friendpost – публікація запитання щодо переїзду в нове місто, до якої віднесено цю сферу;

– jobsphere – сфера роботи, до якої відноситься публікація запитання щодо переїзду в нове місто.

Friendpostjobsphere необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Jobsphere і Friendpost, що виник через потребу забезпечити віднесення публікації запитання користувача щодо переїзду в нове місто до різних професій, а кожній професії забезпечити можливість багаторазового вибору для різних публікацій запитання користувача щодо переїзду в нове місто.

Відповіді користувачів на публікації запитань щодо переїзду в нове місто визначені таблицею Friendpostanswer з рядом полів:

– id – ідентифікатор відповіді користувача, який може бути потенційним другом, на публікацію запитання щодо переїзду в нове місто;

– friendpost – публікація запитання щодо переїзду в нове місто, на яку цю відповідь було сформовано користувачем, який може бути потенційним другом;

– friendapplicant – користувач, який може бути потенційним другом і який відповів на запитання щодо переїзду в нове місто;

– answerdate – дата події публікації відповіді;

– answertext – текст самої відповіді, яку надав на запитання щодо переїзду в нове місто користувач, який може бути потенційним другом.

Friendpost і Friendpostanswer є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що публікація запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості відповідей на неї, а кожна відповідь відноситься тільки до однієї публікації запитання користувача щодо переїзду в нове місто.

Friendapplicant і Friendpostanswer є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один користувач, який може стати потенційним другом, може опублікувати від 0 до певної кількості відповідей на різні запитання, а кожна відповідь публікується тільки одним автором-користувачем.

Фотографії до відповідей на публікації запитань щодо переїзду в нове місто визначені таблицею Friendpostanswerphoto з рядом полів:

– id – ідентифікатор фотографії до відповіді на публікацію запитання щодо переїзду в нове місто;

– friendpostanswer – відповідь користувача, який може бути потенційним другом, на публікацію запитання щодо переїзду в нове місто, щодо якої надано відповідь;

– photo – вміст фотографії, яка додана до відповіді на публікацію запитання щодо переїзду в нове місто.

Friendpostanswer і Friendpostanswerphoto є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що відповідь на публікацію запитання користувача щодо переїзду в нове місто може мати від 0 до певної кількості фотографій, а одна фотографія – додаток до тільки однієї відповіді на публікацію запитання користувача щодо переїзду в нове місто.

Відзначення корисною відповіді на запитання щодо переїзду в нове місто визначено таблицею Friendpostanswerplus з рядом полів:

- id – ідентифікатор відзначення корисною відповіді на запитання щодо переїзду в нове місто;
- friendapplicant – користувач, який може бути потенційним другом, що відзначив цю відповідь як корисну;
- friendpostanswer – відповідь на запитання щодо переїзду в нове місто, яку відзначив корисною користувач.

Friendpostanswerplus необхідна, щоб забезпечити зв'язок багато-до-багатьох між таблицями Friendapplicant і Friendpostanswer, що виник через потребу забезпечити відзначення корисною відповіді на запитання щодо переїзду в нове місто для різних користувачів, але і одночасну потребу для одного й того самого користувача відзначати корисними різні відповіді на запитання щодо переїзду в нове місто.

Запит на дружбу в додатку між користувачами визначений таблицею Friendrequest з рядом полів:

- id – ідентифікатор запиту на дружбу в додатку між користувачами;
- friendapplicant – користувач, який може бути потенційним другом та до якого звернувся інший користувач з запитом на дружбу в додатку;
- requester – користувач, який звернувся за запитом на дружбу в додатку;
- requestdate – час події створення запиту на дружбу в додатку;
- requestresult – результат опрацювання запиту на дружбу в додатку: цей запит може залишатися без відповіді, мати позитивну або негативну відповідь.

Friendapplicant і Friendrequest є таблицями, зв'язаними за типом один-до-багатьох (при чому двічі) через потребу в тому, що один користувач може мати від 0 до певної кількості створених ним запитів на дружбу, а один запит на дружбу створюється тільки одним користувачем, який може бути потенційним другом. Окрім того один користувач може мати від 0 до певної

кількості адресованих йому запитів на дружбу, а один запит на дружбу адресується тільки одному користувачу, який може бути потенційним другом (рис. 2.2).

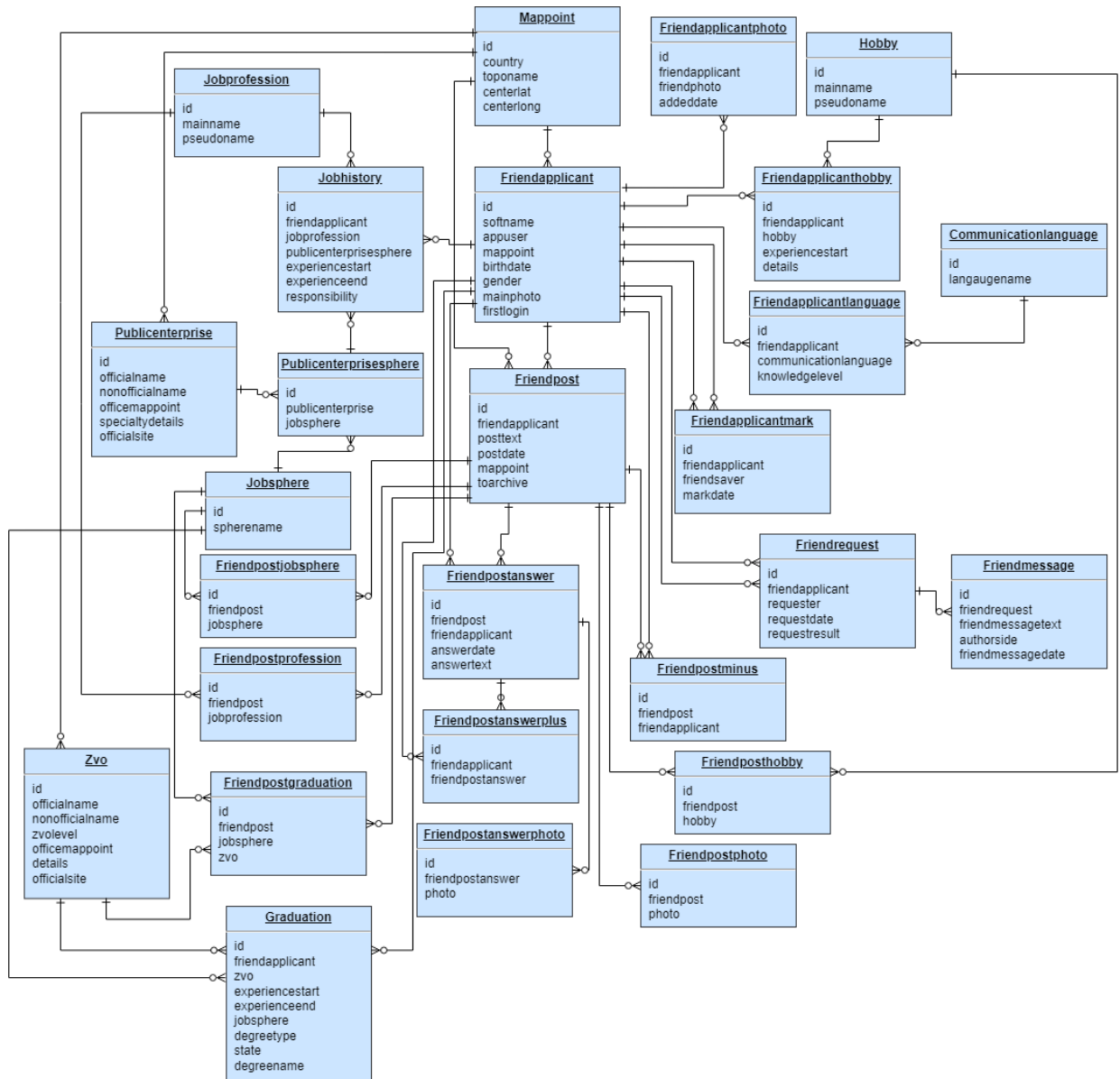


Рисунок 2.2 – Схема БД програми для пошуку друзів у містах при переїзді

Напряму відправлені повідомлення між друзями в додатку визначені таблицею Friendmessage з рядом полів:

– id – ідентифікатор напряму відправленого повідомлення між друзями в додатку;

- friendrequest – запит на дружбу в додатку між користувачами, за яким наряду відправлено повідомлення;
- friendmessagetext – зміст наряду відправленого повідомлення між друзями в додатку;
- authorside – ініціатор запиту на дружбу в додатку, за яким відправлено повідомлення, або адресат запиту є автором самого повідомлення;
- friendmessagedate – час події відправлення наряду повідомлення.

Friendrequest і Friendmessage є таблицями, зв'язаними за типом один-до-багатьох через потребу в тому, що один запит на дружбу може бути використаний для створення від 0 до певної кількості наряду відправлених повідомлень, а кожне повідомлення відноситься тільки до одного запиту на дружбу.

2.4 Висновки за розділом 2

При побудові БД програми для пошуку друзів у містах при переїзді визначено можливості прив'язування запитань щодо переїзду в нове місто до професій, до сфер роботи, до параметрів навчання, до хобі. Усі необхідні сутності перетворені на таблиці реляційної БД на підґрунті використання системи MySQL, що було вирішено інтегрувати до структури додатку на основі Django як ВФ та Python як мови реалізації.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Порядок функціонування програми

Порядок функціонування програми для пошуку друзів у містах при переїзді чітко залежить від порядку вибору дій користувача (рис. 3.1).

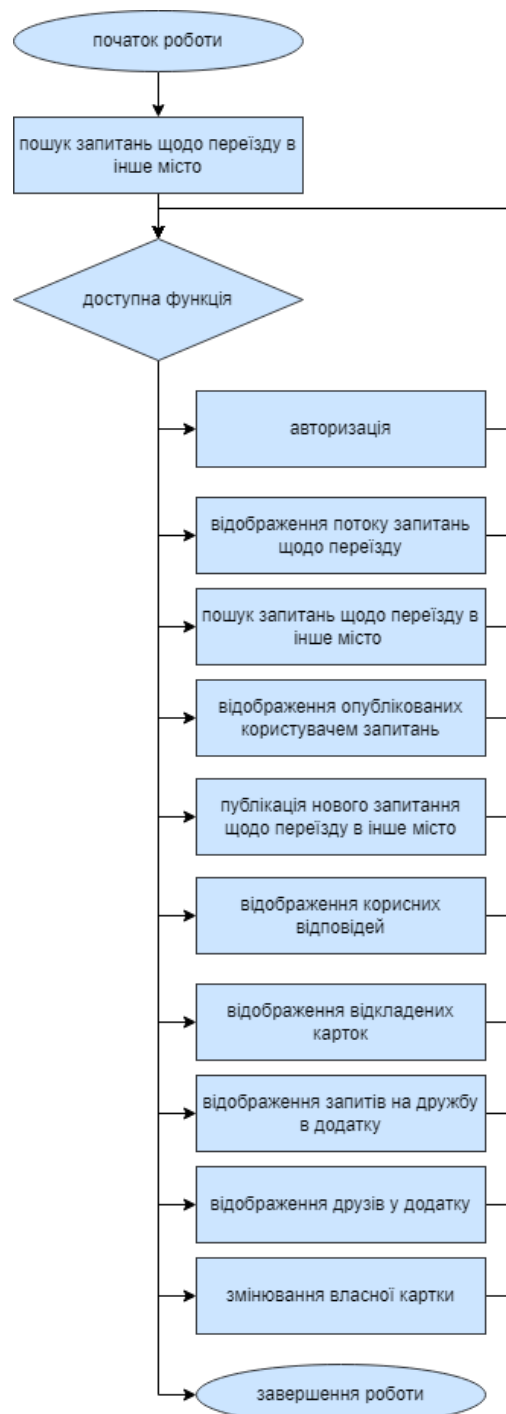


Рисунок 3.1 – Схема функціонування програми для пошуку друзів у містах при переїзді

3.2 Результати реалізації програмних одиниць додатку

Описуючи те, як було реалізовано програмний додаток та які конкретно програмні одиниці були створені, спочатку необхідно вказати на те, що склад програмних одиниць визначається тим, що обрано було ВФ Django для реалізації, а це означає, що моделі та представлення є основними програмними одиницями відносно створення коду. Проте представлення одночасно і реалізують ті функціональні можливості, які необхідно було створити, і використовують для цього підготовлені шаблони відповідних сторінок, і звертаються до моделей, щоби з них відібрати потрібні дані. Тому опис отриманих результатів реалізації розпочато саме з представлень, а далі приведені створені моделі.

Реалізацію формування потоку користувача з запитань щодо переїзду виконано функцією `friendpost_feed_view`. Потік може бути сформований тільки тоді, коли користувач авторизувався, тому у всіх інших випадках відображається сторінка з описом помилки. Якщо авторизація активна, то з картки користувача визначається місто, звідки він працює в програмі, і за цими даними відбирається набір публікацій всіх запитань щодо переїзду в це місто. Якщо користувач скасував для себе відображення запитання, яке входить в число відібраних, то воно відкидається з результатів в потоці. За кожною публікацією запитання щодо переїзду в нове місто визначаються фотографії, інші пов'язані дані, а ось відповіді не включаються, але розраховується кількість отриманих відповідей на кожне запитання. Також за кожним користувачем, що є автором публікації запитання щодо переїзду в нове місто, перевіряється, чи було до цього користувача подано запит на дружбу в додатку та в якому стані цей запит. Усі дані наповнюють шаблон `qfeed.html`, який разом з цими даними і є результатом роботи функції.

Реалізацію створення відповіді на запитання щодо переїзду виконано функцією `friendpost_answer_create_view`, при чому заповнення моделі даними

відповіді виконується через форму `FriendpostanswerForm`, яка відповідає структурі моделі `Friendpostanswer`. Автор відповіді вноситься програмно.

Реалізацію пошуку запитань користувачів щодо переїзду виконано функцією `friendpost_search_view`. Вона отримує цілий ряд параметрів пошуку, проте ці параметри не є окремими параметрами функції, а всі витягаються з єдиного параметра `request`, де за допомогою атрибута `GET` отримується словник зі значеннями параметрів `GET`-запиту. У результатах залишаються запитання користувачів щодо переїзду, які не були заархівовані, відповідають за значенням ключа `point` заданому номеру міста, про переїзд в яке йдеться в запитанні, були опубліковані протягом проміжку дат з `day1.month1.year1` до `day2.month2.year2`, відносяться до хобі з номером `hobby`, до професії з номером `prof`, до сфери роботи з номером `sphere`, до закладу освіти з номером `zvo`.

Реалізацію відображення списку опублікованих користувачем запитань щодо переїзду виконано функцією `applicant_friendposts_view`. Параметр `friendapplicant_name` задає логін користувача, опубліковані яким запитання потрібно відобразити. Якщо це значення не задано, то результати формуються в шаблоні сторінки `ownposts.html` для авторизованого користувача.

Реалізацію публікації запитання щодо переїзду виконано функцією `friendpost_create_view`, що працює на основі ряду форм, які відповідають пов'язаним моделям.

Реалізацію відображення картки користувача виконано функцією `friendapplicant_card_view`. Картка визначається логіном користувача, що або задається параметром `friendapplicant_name` функції, або визначається даними авторизації. До картки окрім загальних даних користувача вносяться також дані про його досвід роботи, фотографії, результати навчання, хобі, мови комунікації, інформація про те, чи була картка цього користувача авторизованим користувачем відкладена, чи було подано запит на дружбу в додатку до цього користувача авторизованим користувачем. Також у дані передаються всі опубліковані користувачем запитання щодо переїзду в нове

місто, відповіді, які він надав на запитання інших користувачів щодо переїзду в нове місто.

Реалізацію визначення даних картки користувача виконано функцією `friendapplicant_define_view`, що працює на основі ряду форм, які відповідають пов'язаним моделям.

Реалізацію відображення публікації запитання щодо переїзду в нове місто виконано функцією `friendpost_detail_view`. На відміну від попередніх представлень тут вже до даних самого запитання щодо переїзду в нове місто додаються також відповіді на це запитання, а для кожної відповіді визначається, чи до її автора подав запит на дружбу користувач, який авторизувався, кількість користувачів, які цю відповідь визнали корисною. Усі дані наповнюють шаблон `detailpost.html`, який разом з цими даними і є результатом роботи функції.

Реалізацію відкладання картки потенційного друга виконано функцією `friendapplicant_mark_view`. Автор відкладення вноситься програмного за даними авторизації. Параметр `friendapplicant_name` задає логін користувача, якому належить картка. Якщо таку картку вже було відкладено цим користувачем, то повторно дія не здійснюється.

Реалізацію відображення відкладених карток потенційних друзів виконано функцією `friendapplicant_marks_view`. За кожною картою визначається, чи було оформлено запит на дружбу з цим користувачем.

Реалізацію подання запиту на дружбу в додатку з користувачем виконано функцією `friend_request_view`. Параметр `friendapplicant_name` задає логін користувача, до якого створюється запит на дружбу. Авторство запиту визначається програмного за даними авторизації. Якщо такий запит на дружбу в додатку вже створено, то повторно не створюється.

Реалізацію прийняття запиту на дружбу в додатку виконано функцією `friend_request_submit_view`. Параметр `friendrequestid` визначає номер запиту на дружбу в додатку.

Реалізацію відображення нових ініційованих та адресованих запитів на дружбу виконано функцією `applicant_requests_view`. Параметр `addressed` визначає, чи потрібні ініційовані авторизованим користувачем, чи адресовані ним запити на дружбу. Відбираються тільки ті запити на дружбу в додатку, які ще не були оброблені тією стороною, якій ці запити були подані.

Реалізацію відображення друзів користувача в додатку виконано функцією `applicant_friends_view`. Параметром `friendapplicant_name` може бути задано логін користувача, а якщо ні, то беруться дані з авторизації. Виділяються тільки ті користувачі, з якими авторизований або заданий користувач мають підтверджений запит на дружбу в додатку.

Реалізацію відображення відправлених повідомлень між друзями, включаючи відправлення напряду повідомлення другу, виконано функцією `applicant_friend_messages_view`. Параметр `friendapplicant_name` задає логін користувача, повідомлення яким були відправлені авторизованому користувачу або отримані від авторизованого користувача. В результаті шаблон вебсторінки не тільки заповнюється набором повідомлень, але і формою, яка в режимі звернення через POST-запит може надавати дані, до яких додаються дані про авторство і адресата повідомлення, зберігаючи всі ці дані разом як напряду відправлене повідомлення між користувачами.

Реалізацію відображення списку опублікованих користувачем відповідей на запитання щодо переїзду виконано функцією `applicant_answers_view`. Параметр `friendapplicant_name` задає логін користувача, опубліковані яким відповіді потрібно відобразити. Якщо це значення не задано, то результати формуються в шаблоні сторінки `ownanswers.html` для авторизованого користувача. За кожною відповіддю також передається перелік інших відповідей на те саме опубліковане запитання щодо переїзду в нове місто.

Реалізацію відзначення корисною відповіді на запитання щодо переїзду в нове місто виконано функцією `friendpost_answer_plus_view`, де параметр

`friendpostanswer` задає номер відповіді на запитання. Раніше створене відзначення вибуває.

Реалізацію відображення списку корисних для користувача відповідей виконано функцією `applicant_useful_answers_view`. Відповіді сортуються за часом відповіді, до набору входять тільки ті, які були визначені авторизованим користувачем як корисні. Додатково до відповіді також додаються всі інші відповіді на те саме опубліковане запитання щодо переїзду в інше місто, а також визначається щодо авторів те, чи було подано авторизованим користувачем до них запит на дружбу та в якому саме він стані, якщо подано.

Реалізацію відхилення запиту на дружбу в додатку виконано функцією `friend_request_cancel_view`. Параметр `friendrequestid` визначає номер запиту на дружбу в додатку.

Реалізацію видалення відкладеної картки потенційного друга виконано функцією `friendapplicant_mark_delete_view`. Параметр `friendapplicantmarkid` задає номер відкладення картки.

Реалізацію закриття запитання щодо переїзду виконано функцією `friendpost_archive_view`, де параметр `friendpostid` задає номер опублікованого запитання щодо переїзду, яке по-перше, повинно існувати, а по-друге, належати авторизованому користувачу.

Реалізацію скасування відображення запитання щодо переїзду в нове місто в потоці виконано функцією `friendpost_minus_view`, де параметр `friendpostid` задає номер запитання щодо переїзду в нове місто.

Реалізацію даних міст, куди переїжджають люди та де шукають собі друзів, виконано моделлю `Mappoint`.

Поле `country` моделі `Mappoint` створює країну, в якій знаходиться це місто, використовуючи в основі тип `CharField`.

Поле `toponame` моделі `Mappoint` створює офіційну назву міста, використовуючи в основі тип `CharField`.

Поля `centerlat` та `centerlong` моделі `Mappoint` створюють географічну локацію (широту і довготу) центра міста, використовуючи в основі тип `FloatField`.

Реалізацію даних картки користувачів програми, що можуть бути комусь потенційними або фактичними друзями, проживаючи в цільовому місті, або знаходитись в пошуку друзів або інформації від них у цільовому місті, виконано моделлю `Friendapplicant`.

Поле `softname` моделі `Friendapplicant` створює ім'я користувача, який може бути потенційним другом в програмі, яким він відображається, використовуючи в основі тип `CharField`.

Поле `appuser` моделі `Friendapplicant` створює користувацький запис, за яким цей потенційний друг працює в Django, а відповідно і в програмі, зв'язуючи цю модель з моделлю `models.User` з `django.contrib.auth`.

Поле `mappoint` моделі `Friendapplicant` створює місто, де на даний момент проживає користувач, який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Mappoint`.

Поле `birthdate` моделі `Friendapplicant` створює дату народження користувача, який може бути потенційним другом, використовуючи в основі тип `DateTimeField`.

Поле `gender` моделі `Friendapplicant` створює стать користувача, який може бути потенційним другом, використовуючи в основі тип `BooleanField`.

Поле `mainphoto` моделі `Friendapplicant` створює фотографію, на якій показаний користувач, який може бути потенційним другом, використовуючи в основі тип `ImageField`.

Поле `firstlogin` моделі `Friendapplicant` створює час події першої авторизації користувача, який може бути потенційним другом, в програмі, використовуючи в основі тип `DateTimeField`.

Реалізацію даних додаткових фотографій користувачів, які можуть бути потенційними друзями, виконано моделлю `Friendapplicantphoto`.

Поле `friendapplicant` моделі `Friendapplicantphoto` створює користувача, який може бути потенційним другом і який зображений на фотографії, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `friendphoto` моделі `Friendapplicantphoto` створює саму фотографію, на якій показаний користувач, який може бути потенційним другом, використовуючи в основі тип `ImageField`.

Поле `addeddate` моделі `Friendapplicantphoto` створює час події додавання фотографії в програму, використовуючи в основі тип `DateTimeField`.

Реалізацію даних професії, що можуть задаватися в програмі, виконано моделлю `Jobprofession`.

Поле `mainname` моделі `Jobprofession` створює основну назву професії, використовуючи в основі тип `CharField`.

Поле `pseudoname` моделі `Jobprofession` створює додаткову назву професії (за наявності), використовуючи в основі тип `CharField`.

Реалізацію даних історії досвіду роботи користувача, який може бути потенційним другом, виконано моделлю `Jobhistory`.

Поле `friendapplicant` моделі `Jobhistory` створює користувача, для якого цей досвід роботи визначений та який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `jobprofession` моделі `Jobhistory` створює професію, за якою працював на цьому підприємстві цей користувач, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Jobprofession`.

Поле `publicenterprisesphere` моделі `Jobhistory` створює сферу роботи підприємства-роботодавця, до якої відносилась виконував робота користувачем, який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Publicenterprisesphere`.

Поле `experiencestart` моделі `Jobhistory` створює дату події влаштування на цю роботу, використовуючи в основі тип `DateField`.

Поле `experienceend` моделі `Jobhistory` створює дату події звільнення з цієї роботи, використовуючи в основі тип `DateField`.

Поле `responsibility` моделі `Jobhistory` створює опис особливостей робочих обов'язків, які виконував працівник, використовуючи в основі тип `CharField`.

Реалізацію даних сфери роботи виконано моделлю `Jobsphere`, в якій полем `spherename` створено назву сфери роботи, використовуючи в основі тип `CharField`.

Реалізацію даних підприємства-роботодавця виконано моделлю `Publicenterprise`.

Поле `officialname` моделі `Publicenterprise` створює офіційну назву підприємства-роботодавця, використовуючи в основі тип `CharField`.

Поле `nonofficialname` моделі `Publicenterprise` створює неофіційну назву підприємства-роботодавця, використовуючи в основі тип `CharField`.

Поле `officemapoint` моделі `Publicenterprise` створює місто, де знаходиться головний офіс підприємства, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Mapoint`.

Поле `specialtydetails` моделі `Publicenterprise` створює особливості роботи підприємства-роботодавця, надаваних послуг, використовуючи в основі тип `CharField`.

Поле `officialsite` моделі `Publicenterprise` створює офіційний сайт підприємства-роботодавця, використовуючи в основі тип `URLField`.

Реалізацію даних сфер, в яких працюють підприємства-роботодавці, виконано моделлю `Publicenterprisesphere`.

Поле `publicenterprise` моделі `Publicenterprisesphere` створює підприємство-роботодавця, що працює в цій сфері, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Publicenterprise`.

Поле `jobsphere` моделі `Publicenterprisesphere` створює сферу, в якій працює підприємство-роботодавець, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Jobsphere`.

Реалізацію даних хобі, які для себе можуть визначати користувачі, що можуть бути потенційними друзями, виконано моделлю `Hobby`.

Поле `mainname` моделі `Hobby` створює основну назву хобі, використовуючи в основі тип `CharField`.

Поле `pseudoname` моделі `Hobby` створює додаткову назву хобі, використовуючи в основі тип `CharField`.

Реалізацію даних хобі користувачів, які можуть бути потенційними друзями, виконано моделлю `Friendapplicanthobby`.

Поле `friendapplicant` моделі `Friendapplicanthobby` створює користувач, який може бути потенційним другом та який займається цим хобі, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `hobby` моделі `Friendapplicanthobby` створює хобі, яким займається користувач, який є потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Hobby`.

Поле `experiencestart` моделі `Friendapplicanthobby` створює час події, коли вперше користувач, який може бути потенційним другом, зайнявся цим хобі, використовуючи в основі тип `DateTimeField`.

Поле `details` моделі `Friendapplicanthobby` створює особливості досвіду заняття цим хобі користувачем, який може бути потенційним другом, використовуючи в основі тип `CharField`.

Реалізацію даних закладів освіти, доступних в програмі, виконано моделлю `Zvo`.

Поле `officialname` моделі `Zvo` створює офіційну назву закладу освіти, використовуючи в основі тип `CharField`.

Поле `zvolevel` моделі `Zvo` створює рівень закладу освіти, використовуючи в основі тип `PositiveSmallIntegerField`.

Поле `nonofficialname` моделі `Zvo` створює неофіційну назву закладу освіти або колишню назву закладу освіти, використовуючи в основі тип `CharField`.

Поле `officemapoint` моделі `Zvo` створює місто, де знаходиться головний офіс закладу освіти, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Mappoint`.

Поле `details` моделі `Zvo` створює особливості навчання, організації роботи закладу освіти, використовуючи в основі тип `CharField`.

Поле `officialsite` моделі `Zvo` створює офіційний вебсайт закладу освіти, використовуючи в основі тип `URLField`.

Реалізацію даних результатів навчання користувачів, які можуть бути потенційними друзями, виконано моделлю `Graduation`.

Поле `friendapplicant` моделі `Graduation` створює користувач, який може бути потенційним другом і який навчався за цією програмою, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `zvo` моделі `Graduation` створює заклад освіти, де навчався користувач, який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Zvo`.

Поле `experiencestart` моделі `Graduation` створює дата події вступу до закладу освіти, використовуючи в основі тип `DateField`.

Поле `experienceend` моделі `Graduation` створює дата події завершення навчання в закладі освіти, використовуючи в основі тип `DateField`.

Поле `jobsphere` моделі `Graduation` створює сферу роботи, за якою навчання відбувалось, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Jobsphere`.

Поле `degreetype` моделі `Graduation` створює рівень освіти, за яким здійснювалось навчання, використовуючи в основі тип `CharField`.

Поле `state` моделі `Graduation` створює стан навчання, використовуючи в основі тип `CharField`.

Поле `degreename` моделі `Graduation` створює назву отриманої кваліфікації, використовуючи в основі тип `CharField`.

Реалізацію даних доступних для вибору в програмі мови для комунікації виконано моделлю `Communicationlanguage` з полем `langaugename` назви мови для комунікації, реалізованого типом `CharField`.

Реалізацію даних мов, якими володіють користувачі, які можуть бути потенційними друзями, виконано моделлю `Friendapplicantlanguage`.

Поле `friendapplicant` моделі `Friendapplicantlanguage` створює користувача, який може бути потенційним другом і який володіє цією мовою, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `communicationlanguage` моделі `Friendapplicantlanguage` створює мову для комунікації, якою володіє цей користувач, який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Communicationlanguage`.

Поле `knowledgelevel` моделі `Friendapplicantlanguage` створює рівень знання цієї мови для комунікації користувачем, який може бути потенційним другом, використовуючи в основі тип `CharField`.

Реалізацію даних відкладання карток потенційних друзів виконано моделлю `Friendapplicantmark`.

Поле `friendapplicant` моделі `Friendapplicantmark` створює картку потенційного друга, яку відклав користувач з наміром переглянути її пізніше, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `friendsaver` моделі `Friendapplicantmark` створює картку користувача, який відклав цю картку потенційного друга, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `markdate` моделі `Friendapplicantmark` створює час події відкладання картки потенційного друга, використовуючи в основі тип `DateTimeField`.

Реалізацію даних публікації запитань користувачів щодо переїзду в нове місто виконано моделлю Friendpost.

Поле friendapplicant моделі Friendpost створює користувача, який може бути потенційним другом та який поставив певне запитання стосовно переїзду в нове місто, використовуючи в основі тип ForeignKey для зв'язування з моделлю Friendapplicant.

Поле posttext моделі Friendpost створює текст запитання стосовно переїзду в нове місто, використовуючи в основі тип CharField.

Поле toarchive моделі Friendpost створює позначку, що публікація запитання користувача щодо переїзду в нове місто має бути заархівована, використовуючи в основі тип BooleanField.

Поле postdate моделі Friendpost створює дату події публікації запитання користувачем щодо переїзду в нове місто, використовуючи в основі тип DateTimeField.

Поле mappoint моделі Friendpost створює місто, куди має намір переїхати користувач, який може бути потенційним другом, використовуючи в основі тип ForeignKey для зв'язування з моделлю Mappoint.

Поле minuses створює перелік користувачів, які можуть бути потенційними друзями та які скасували відображення для себе цього запитання щодо переїзду в нове місто іншого користувача, використовуючи в основі тип ManyToManyField для зв'язування з моделлю Friendapplicant для забезпечення зв'язку багато-до-багатьох.

Поле hobbies створює перелік хобі, які віднесені до цієї публікації запитання користувача щодо переїзду в нове місто, використовуючи в основі тип ManyToManyField для зв'язування з моделлю Hobby для забезпечення зв'язку багато-до-багатьох.

Поле jobprofessions створює перелік професій, які віднесені до цієї публікації запитання користувача щодо переїзду в нове місто, використовуючи в основі тип ManyToManyField для зв'язування з моделлю Jobprofession для забезпечення зв'язку багато-до-багатьох.

Поле `jobspheres` створює перелік сфер роботи, які віднесені до цієї публікації запитання користувача щодо переїзду в нове місто, використовуючи в основі тип `ManyToManyField` для зв'язування з моделлю `Jobsphere` для забезпечення зв'язку багато-до-багатьох.

Реалізацію даних фотографії з публікацій запитань щодо переїзду в нове місто виконано моделлю `Friendpostphoto`.

Поле `friendpost` моделі `Friendpostphoto` створює публікацію запитання користувача щодо переїзду в нове місто, до якої фотографія віднесена, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendpost`.

Поле `photo` моделі `Friendpostphoto` створює вміст фотографії, що відноситься до цієї публікації запитання користувача щодо переїзду в нове місто, використовуючи в основі тип `ImageField`.

Реалізацію даних віднесення запитання щодо переїзду до навчання виконано моделлю `Friendpostgraduation`.

Поле `friendpost` моделі `Friendpostgraduation` створює публікацію запитання щодо переїзду в нове місто, до якої віднесено ці параметри навчання, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendpost`.

Поле `jobsphere` моделі `Friendpostgraduation` створює сферу роботи, за якою здійснюється навчання, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Jobsphere`.

Поле `zvo` моделі `Friendpostgraduation` створює заклад освіти, стосовно якого поставлено запитання щодо переїзду в нове місто, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Zvo`.

Реалізацію даних відповіді користувачів на публікації запитань щодо переїзду в нове місто виконано моделлю `Friendpostanswer`.

Поле `friendpost` моделі `Friendpostanswer` створює публікацію запитання щодо переїзду в нове місто, на яку цю відповідь було сформовано

користувачем, який може бути потенційним другом, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendpost`.

Поле `friendapplicant` моделі `Friendpostanswer` створює користувача, який може бути потенційним другом і який відповів на запитання щодо переїзду в нове місто, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `answerdate` моделі `Friendpostanswer` створює дату події публікації відповіді, використовуючи в основі тип `DateTimeField`.

Поле `answertext` моделі `Friendpostanswer` створює текст самої відповіді, яку надав на запитання щодо переїзду в нове місто користувач, який може бути потенційним другом, використовуючи в основі тип `CharField`.

Реалізацію даних фотографії до відповідей на публікації запитань щодо переїзду в нове місто виконано моделлю `Friendpostanswerphoto`.

Поле `friendpostanswer` моделі `Friendpostanswerphoto` створює відповідь користувача, який може бути потенційним другом, на публікацію запитання щодо переїзду в нове місто, щодо якої надано відповідь, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendpostanswer`.

Поле `photo` моделі `Friendpostanswerphoto` створює вміст фотографії, яка додана до відповіді на публікацію запитання щодо переїзду в нове місто, використовуючи в основі тип `ImageField`.

Поле `pluses` створює перелік користувачів, які відзначили корисною цю відповідь на публікацію запитання щодо переїзду в нове місто, використовуючи в основі тип `ManyToManyField` для зв'язування з моделлю `Friendpostanswer` для забезпечення зв'язку багато-до-багатьох.

Реалізацію даних запиту на дружбу в додатку між користувачами виконано моделлю `Friendrequest`.

Поле `friendapplicant` моделі `Friendrequest` створює користувача, який може бути потенційним другом та до якого звернувся інший користувач з запитом на дружбу в додатку, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `requester` моделі `Friendrequest` створює користувача, який звернувся за запитом на дружбу в додатку, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendapplicant`.

Поле `requestdate` моделі `Friendrequest` створює час події створення запиту на дружбу в додатку, використовуючи в основі тип `DateTimeField`.

Поле `requestresult` моделі `Friendrequest` створює результат опрацювання запиту на дружбу в додатку, використовуючи в основі тип `PositiveSmallIntegerField`.

Реалізацію даних напряду відправлених повідомлень між друзями в додатку виконано моделлю `Friendmessage`.

Поле `friendrequest` моделі `Friendmessage` створює запит на дружбу в додатку між користувачами, за яким напряду відправлено повідомлення, використовуючи в основі тип `ForeignKey` для зв'язування з моделлю `Friendrequest`.

Поле `friendmessagetext` моделі `Friendmessage` створює вміст напряду відправленого повідомлення між друзями в додатку, використовуючи в основі тип `CharField`.

Поле `authorside` моделі `Friendmessage` створює ініціатора запиту на дружбу в додатку, за яким відправлено повідомлення, або адресат запиту є автором самого повідомлення, використовуючи в основі тип `BooleanField`.

Поле `friendmessagedate` моделі `Friendmessage` створює час події відправлення повідомлення, використовуючи в основі тип `DateTimeField`.

3.3 Висновки за розділом 3

У розділі описано те, як було реалізовано програмний додаток та які конкретно програмні одиниці були створені. Безпосередньо склад програмних одиниць визначався тим, що обрано було ВФ Django для реалізації, а це означає, що моделі та представлення є основними програмними одиницями відносно створення коду, на чому і було зроблено акцент при описі.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма забезпечує пошук друзів у містах, в які користувачі бажають переїхати, та надає можливість підтримки цього процесу щодо публікації тих запитань, які виникають у користувачів та надання на них відповідей. Відповідно програма може використовуватися з одного боку користувачами для створення запитань щодо переїзду в нове місто, а з іншого – для надання відповідей на ті запитання щодо переїзду в інше місто, які виникли у інших користувачів.

4.2 Умови виконання програми

Обов'язковою умовою експлуатації програми для пошуку друзів у містах при переїзді є наявність адміністратора програми або користувача, здатного виконувати такі функції. В обов'язки адміністратора входить наповнення даними програми зокрема стосовно міст, сфер роботи, університетів, підприємств-роботодавців тощо, а також керування будь-якими сутностями і даними, які містяться в програмі та які можуть потребувати внесення змін. Також адміністратор повинен виконати розгортання додатку на сервері.

Технічне забезпечення сервера повинно включати мінімально процесор Intel Xeon E5440 (частота 2,83 ГГц), жорсткий диск – 100 Гб (наявність фотографій може значно збільшувати вимоги до сервера, тому це потребує контролю та можливо введення обмежень на обсяг фотографій, які користувач може завантажувати, інакше необхідно розширювати потрібний простір), оперативну пам'ять – 4 Гб.

4.3 Виконання і тестування програми

Програма має основний засіб керування роботою з функціями, виражений меню, що відображається у лівій панелі кожної сторінки, хоча може бути приховано шляхом натискання на іконку з трьома паралельними рисками.

Меню в лівій панелі містить ряд засобів, які будуть перелічені далі (рис. 4.1). У той же час згори сторінки у правому верхньому куті після авторизації користувача з'являється фотографія цього користувача, ім'я та випадаючий перелік, з якого можна обрати перехід до картки самого користувача, звідки можна змінити за необхідності і дані профілю користувача.

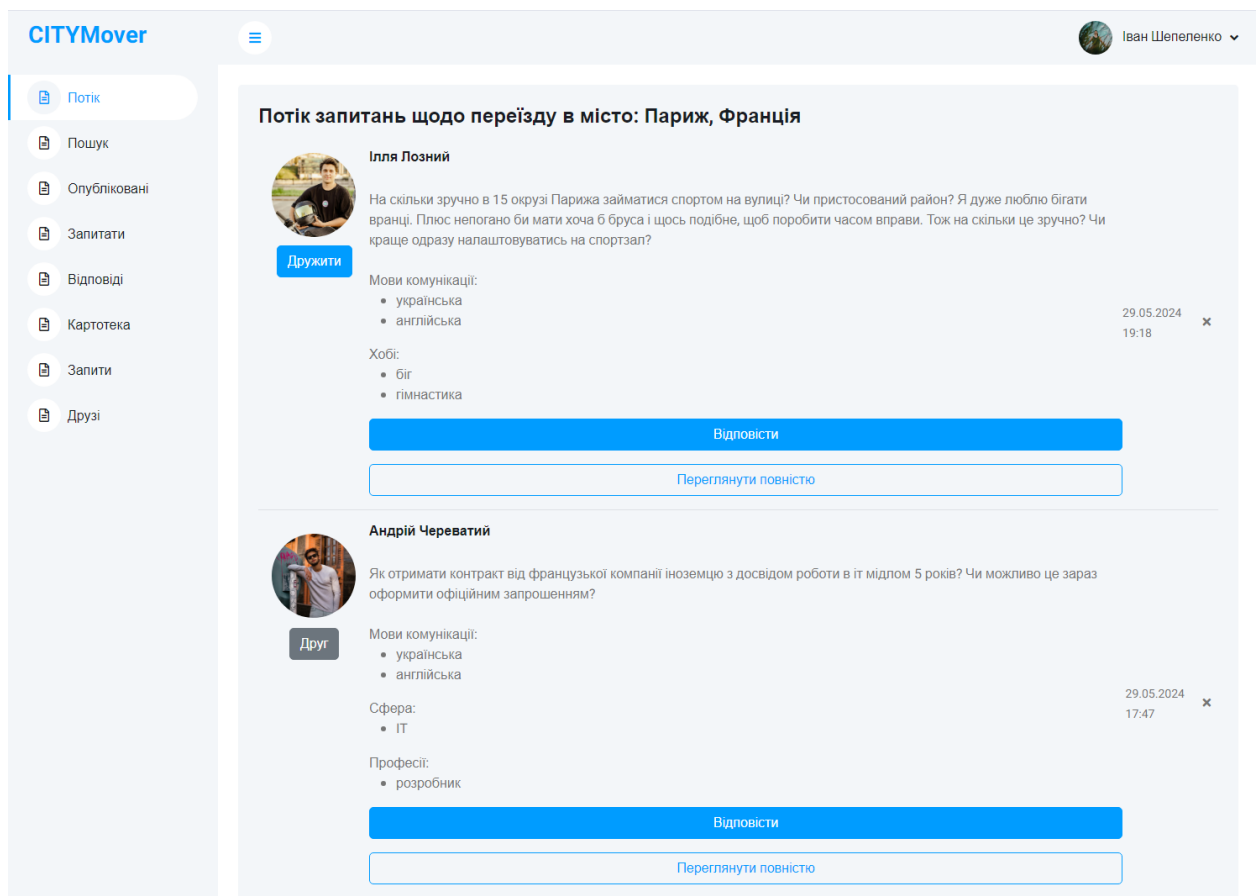


Рисунок 4.1 – Вебсторінка потоку запитань щодо переїзду в місто проживання користувача

Основним засобом щодо отримання інформації про проживання в місті, куди бажає переїхати людина, є формування запитань. Сформувавши запитання можна, обираючи з меню засіб «Запитати», після чого сформувавши запитання. З іншого боку для того, щоб користувачі, які проживають в містах, про які сформовані запитання, могли знайти таке запитання та дати відповідь базовим засобом слід вважати функцію, доступну з меню як «Потік».

Потік запитань щодо переїзду в місто проживання користувача відображає запитання, які стосуються того міста, де проживає авторизований користувач у момент перегляду. Біля кожного запитання показується його автор, а під його фотографією знизу кнопка:

- напис на кнопці «Дружити» означає, що авторизований користувач ще не є з автором запитання другом, але можна подати запит, натиснувши на кнопку;

- напис на кнопці «Друг» означає, що авторизований користувач вже є другом з автором запитання, але можна вилучити його з друзів, натиснувши на кнопку;

- напис на кнопці «Скасувати» означає, що був поданий запит на дружбу, але на нього не було отримано відповідь;

- напис на кнопці «Закрито» означає, що раніше був поданий запит на дружбу, але на нього було дано негативну відповідь.

Натискаючи на фотографію користувача, можна відкрити його картку, де наводиться інформація про нього, а якщо це користувач, який є другом авторизованому, то і можливість написати йому повідомлення.

З потоку запитань кожне запитання містить текст, дату і час публікації, а також пов'язану інформацію про хобі, професії, освіту. Якщо не приведено кількість відповідей, то це означає, що на запитання не було надано відповідей. Якщо були, то вказується тільки кількість. Щоб переглянути всі відповіді до запитання разом з самим запитанням, потрібно відкрити окрему сторінку, що можна зробити, натиснувши на кнопку «Переглянути повністю».

Основним напрямком роботи з кожним запитанням є або дати відповідь на нього, для чого можна одразу натиснути з потоку кнопку «Відповісти» біля відповідного запитання, або скасувати відображення цього запитання щодо переїзду в нове місто, для чого потрібно натиснути на хрестик справа від запитання. У таких випадках запитання більше не буде відображатися. Всі запитання, з якими такі дії не були виконані, відображаються, але нові відображаються вище в потоці.

Картка користувача містить основну інформацію про нього, фотографії (можна перемикатися між ними), інформацію про досвід роботи, навчання, хобі, мови комунікації (рис. 4.2).

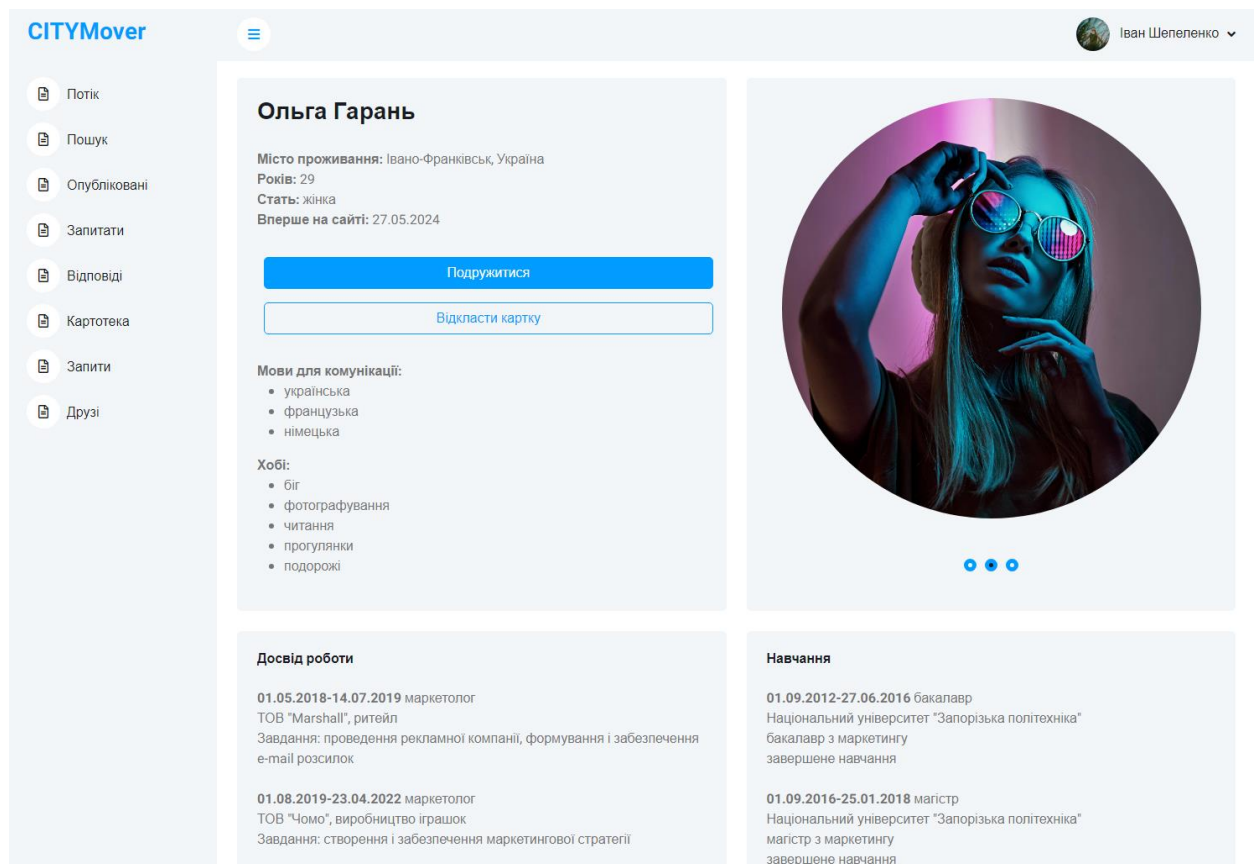


Рисунок 4.2 – Вебсторінка картки користувача, який може бути потенційним другом

З власної картки всі ці дані можна змінювати. Якщо користувач, який авторизувався, є другом для користувача, про якого ця картка, то

відображається кнопка «Написати повідомлення», що дозволяє перейти до спілкування напряму. Якщо це не друг, то відображається кнопка «Подружитися». Звідси також можна скасувати дружбу. На сторінці картки користувача також відображаються ті запитання про переїзд в інше місто, які поставив цей користувач, а також ті відповіді, які він встиг дати на запитання інших користувачів.

Функція «Пошук» з меню зліва дозволяє виконувати пошук серед тих запитань, які поставили інші користувачі, задаючи місто, період дат, коли запитання були поставлені, сферу, професію тощо. Це ще один варіант знайти запитання, на яке можна дати відповідь.

Переглянути свої власні опубліковані запитання щодо переїзду в інше місто можна, вибираючи з меню пункт «Опубліковані». Переглянути корисні для авторизованого користувача відповіді на запитання можна за функцією «Відповіді». Визнати відповідь корисною можна зі сторінки перегляду запитання щодо переїзду в нове місто, де разом з запитанням демонструються і всі відповіді на нього.

Функція меню «Запити» відкриває сторінку з запитами на дружбу в додатку (рис. 4.3). Сторінка працює в двох режимах: якщо натиснуто на кнопку «Вхідні», то відображаються ті запити, які адресовані авторизованому користувачу, а якщо «Вихідні», то ті, які він сам створив.

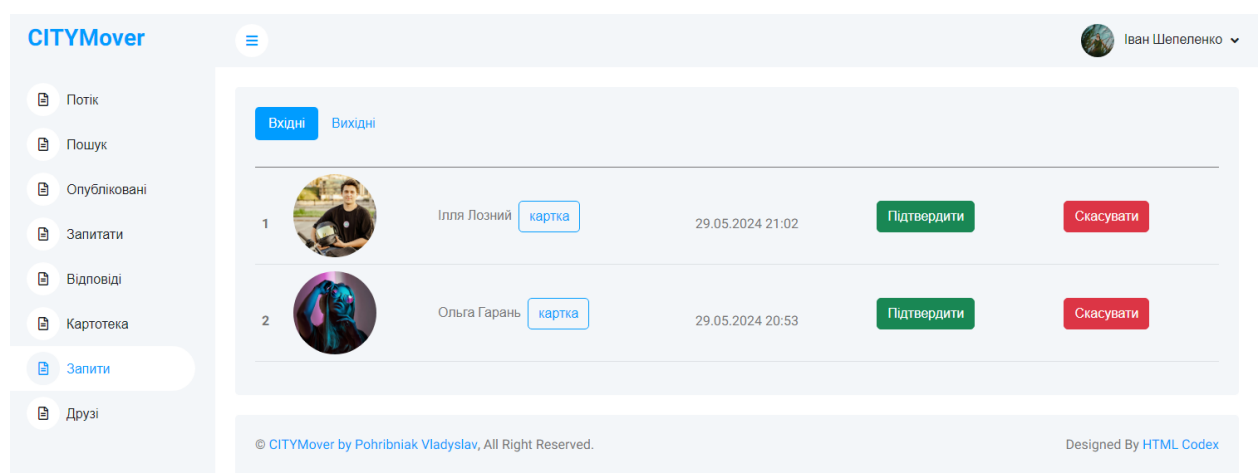


Рисунок 4.3 – Вебсторінка запитів на дружбу в додатку

За кожним вхідним запитом може його або підтвердити, або скасувати відповідними кнопками біля кожного запиту. Після підтвердження запит з'являється на сторінці за функцією «Друзі». Там відображається перелік всіх друзів авторизованого користувача, а обираючи одного з них, можна відкрити перелік всіх відправлених напряму один одному повідомлень.

Дизайн вебсторінок реалізований за безкоштовним шаблоном [7] (рис. 4.3).

Функція меню «Картотека» відображає всі відкладені авторизованим користувачем картки інших користувачів.

Верхнє меню до виконання авторизації містить кнопку для виконання такої авторизації, а після її виконання з випадаючого переліку можна обрати вихід.

Тестування всіх функцій було реалізовано. Для кожної функції було визначено очікуваний результат в різних варіантах. Зокрема у випадку завдання некоректних даних. Далі відбувалась перевірка отриманого результату з встановленим. У підсумку всі функції були зведені до отримання встановлених результатів. Тестування, здійснено через звернення користувача до сервера, виявилось повністю успішним.

4.4 Обробка збійних ситуацій

Як було описано вище, для функцій додатку передбачено обробку отримання некоректних результатів. Для цього використовується одна вебсторінка, але вона містить запис про отриману проблему, що дозволяє ідентифікувати, що саме сталося. Наприклад, це може бути звернення за інформацією за логіном користувача, якого не існує (рис. 4.4). Можливими є також проблеми з авторизацією, коли відбувається звернення до функції, яка вимагає спочатку авторизації. Також можливо відбувається спроба витягнути той чи інший об'єкт, якого не існує (наприклад, задано некоректний номер).

Тоді також виводиться відповідне інформаційне повідомлення, яке дозволяє зрозуміти, що саме було задано некоректно.

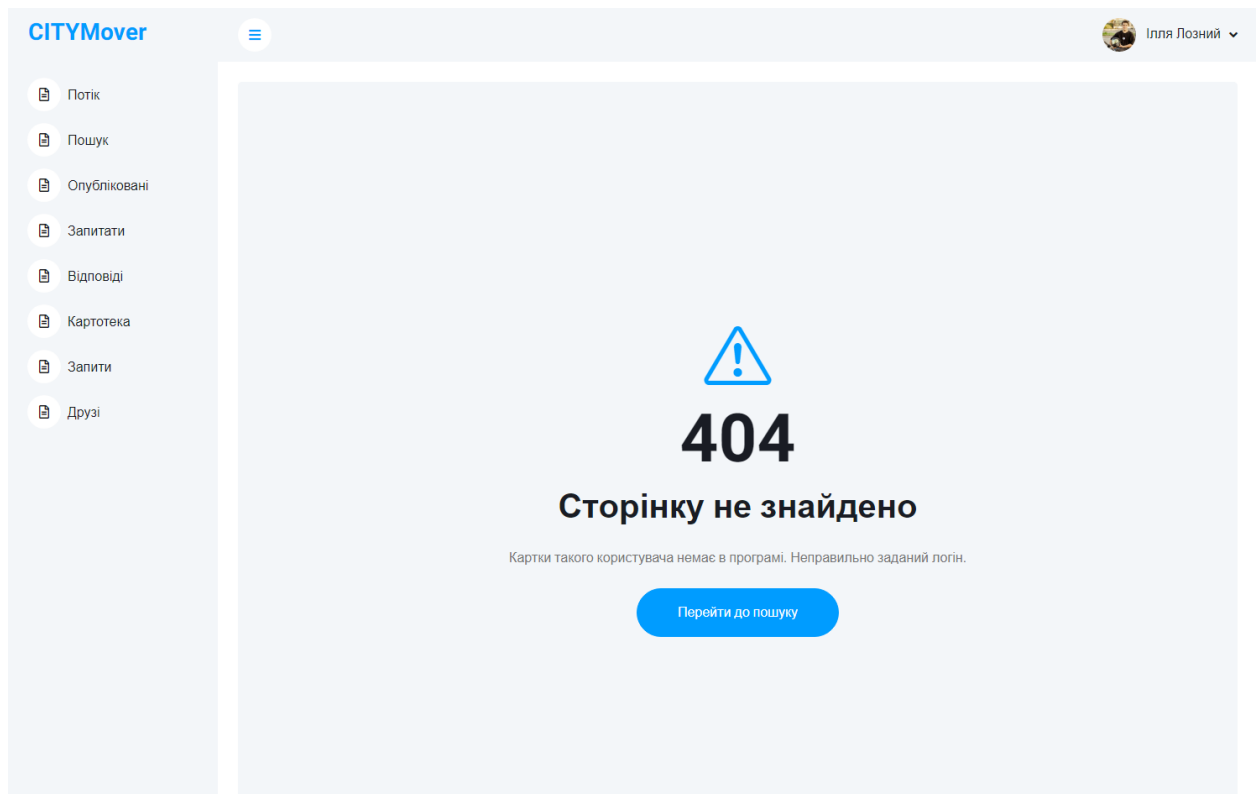


Рисунок 4.4 – Вебсторінка відображення інформації про помилку

4.5 Висновки за розділом 4

Програмне забезпечення для пошуку друзів у містах при переїзді має всі функції реалізовані, що підтверджено успішним тестуванням як у випадках, коли мав бути отриманий певний позитивний результат, так і у випадках, які за заданими вхідними даними є помилковими, – тоді виводяться конкретні повідомлення про виявлені помилки.

ВИСНОВКИ

У роботі розроблено програмне забезпечення для пошуку друзів у містах при переїзді, що дозволило забезпечити підтримку інформування людей під час даного процесу.

Виконаний огляд програмних інструментів, придатних для пошуку друзів у містах при переїзді, встановив, що такі існуючі інструменти, як LinkedIn, Facebook мають певні окремі можливості, які сприяють пошуку друзів або інформації при переїзді в інше місто, проте не є цілеспрямованими для досягнення цього. Тому виділення корисних особливостей з цих програм та об'єднання їх з концепцією запитань є важливим для розв'язання задачі пошуку друзів у містах при переїзді.

Розробку програми виконано базовим набором засобів, що включає мову Python, систему MySQL, фреймворк Django та інші додаткові засоби.

Основним результатом роботи є програма, яка забезпечує пошук друзів у містах, в які користувачі бажають переїхати, та надає можливість підтримки цього процесу щодо публікації тих запитань, які виникають у користувачів та надання на них відповідей. Реалізовано можливості прив'язування запитань щодо переїзду в нове місто до професій, до сфер роботи, до параметрів навчання, до хобі. Створено потік запитань щодо переїзду в місто, де проживає авторизований користувач і щодо якого він може розглядатися як експерт, з можливістю відкидання запитань або створення відповідей на них.

Розробку програми виконано через забезпечення етапів, що охоплювали визначення особливостей аналогів і порівняння з розробкою, виділення основних засобів проєктування, реалізацію моделей, представлень, шаблонів для забезпечення всіх функціональних можливостей. Усі визначені завдання та безпосередньо сама розробка успішно завершені.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. LinkedIn Україна [Електрон. ресурс]. – Режим доступу : <https://ua.linkedin.com/>.
2. Facebook [Electronic resource]. – Access mode : <https://www.facebook.com/>.
3. Installation – Laravel 11.x – The PHP Framework For Web Artisans [Electronic resource]. – Access mode : <https://laravel.com/docs/11.x>.
4. GitHub – laravel/laravel [Electronic resource] : Laravel is a web application framework with expressive, elegant syntax. – Access mode : <https://github.com/laravel/laravel>.
5. What is Django? – Django Framework Explained – AWS [Electronic resource]. – Access mode : <https://aws.amazon.com/what-is/django/>.
6. Getting Started with Django Tutorial [Electronic resource]. – Access mode : <https://learndjango.com/tutorials/getting-started-django-tutorial>.
7. DASHMIN – Bootstrap Admin Template Free [Electronic resource]. – Access mode : <https://htmlcodex.com/bootstrap-admin-template-free/>.

ДОДАТОК А
Технічне завдання

Вступ

Багато українців вимушено відчуло на собі проблеми, пов'язані з переїздом в інші міста. Знайти нове коло спілкування може бути додатковою проблемою, однак отримати початкову інформацію, зрозуміти, наскільки дане місто взагалі підходить для способу життя людини і чи не обрати інше, є дуже важливими запитаннями. Для відповідей на такі запитання є дуже корисне джерело – люди, які в цьому місті проживають.

Однак запитання можуть бути спеціалізованими: стосуватися навчання, роботи за певними професіями або роботи загалом, певних хобі (як наприклад, готовності міського простору для занять бігом на вулицях міста). У таких випадках важливо віднайти цільову аудиторію таких запитань, які здатні дати потрібні відповіді. Така орієнтованість дозволяє створити певні спільноти в програмі, звертаючись до відповідного адресата та запрошуючи його дати відповідь, а за рахунок привертання учасників таких спільнот (наприклад, ІТ-спільноти) отримувати корисні відповіді, а в подальшому можливо встановлювати більш тісний контакт, знаходити друзів.

Усе це сукупно повинно дозволити забезпечити цілісну підтримку людей під час переїзду в нове місто, в тому числі під час формування таких планів на переїзд. Звичайно, що в сучасному світі це потребує створення програмного інструменту для пошуку друзів у містах при переїзді.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Програмне забезпечення для пошуку друзів у містах при переїзді».

A.2 Призначення розробки

Програма має забезпечити пошуку друзів у містах, в які користувачі бажають переїхати, та надати можливість підтримки цього процесу щодо публікації тих запитань, які виникають у користувачів та надання на них відповідей.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програми пошуку друзів у містах при переїзді повинні включати:

- формування потоку користувача з запитань щодо переїзду;
- скасування відображення запитання щодо переїзду в нове місто в потоці;
- створення відповіді на запитання щодо переїзду;
- пошук запитань користувачів щодо переїзду;
- закриття запитання щодо переїзду;
- відображення списку опублікованих користувачем запитань щодо переїзду;
- публікація запитання щодо переїзду;
- відображення картки користувача;
- визначення даних картки користувача, включаючи визначення досвіду роботи користувача, визначення його хобі, результатів навчання, мов для комунікації;
- авторизація і реєстрація;
- відображення публікації запитання щодо переїзду в нове місто;
- відкладання картки потенційного друга;
- видалення відкладеної картки потенційного друга;
- відображення відкладених карток потенційних друзів;

- подання запиту на дружбу в додатку з користувачем;
- відхилення або прийняття запиту на дружбу в додатку;
- відображення нових ініційованих та адресованих запитів на дружбу;
- відображення друзів користувача в додатку;
- відображення відправлених повідомлень між друзями, включаючи відправлення на пряму повідомлення другу;
- відображення списку опублікованих користувачем відповідей на запитання щодо переїзду;
- відзначення корисною відповіді на запитання щодо переїзду в нове місто;
- скасування відзначення корисною відповіді на запитання щодо переїзду в нове місто;
- відображення списку корисних для користувача відповідей.

A.3.2 Умови експлуатації

Обов'язковою умовою експлуатації програми для пошуку друзів у містах при переїзді є наявність адміністратора програми або користувача, здатного виконувати такі функції. В обов'язки адміністратора входить наповнення даними програми зокрема стосовно міст, сфер роботи, університетів, підприємств-роботодавців тощо, а також керування будь-якими сутностями і даними, які містяться в програмі та які можуть потребувати внесення змін. Також адміністратор повинен виконати розгортання додатку на сервері.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічне забезпечення сервера повинно включати мінімально:

- процесор Intel Xeon E5440 (частота 2,83 ГГц);

- жорсткий диск – 100 Гб (наявність фотографій може значно збільшувати вимоги до сервера, тому це потребує контролю та можливо введення обмежень на обсяг фотографій, які користувач може завантажувати, інакше необхідно розширювати потрібний простір);
- оперативна пам'ять – 4 Гб.

A.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 15 квітня, протягом наступних 8 тижнів має бути забезпечено виконання роботи за календарним планом, що містить в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б
Текст програми

```

from .models import Friendpost
from .models import Friendapplicant
from .models import Friendpostphoto
from .models import Friendpostgraduation
from .models import Friendpostanswer
from .models import Friendrequest
from .models import Friendpostanswerphoto
from .models import Friendmessage
from .models import Friendapplicantmark
from django.shortcuts import render
from django.db.models import Q
from .models import Jobhistory
from .models import Friendapplicanthobby
from .models import Graduation
from .models import Friendapplicantlanguage

def friendpost_feed_view(request):
    if request.user.is_active:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
                    {"error": "Ви не внесли дані про себе в
особову картку, тому не можна сформувати потік з запитань щодо
переїзду."})
        else:
            return render(request, "citymover/error.html",
                {"error": "Без авторизації не можна сформувати
потік з запитань щодо переїзду. Авторизуйтеся, будь ласка, і
поверніться до потоку."})
            if not friendapplicant.mappoint:
                return render(request, "citymover/error.html",
                    {"error": "Неможливо сформувати потік з
запитань щодо переїзду, доки ви не задали місто, в якому
проживаєте."})
            city_for_feed = friendapplicant.mappoint
            friendpost_feed =
Friendpost.objects.prefetch_related("minuses")
            friendpost_feed =
friendpost_feed.prefetch_related("hobbies")
            friendpost_feed =
friendpost_feed.prefetch_related("jobprofessions")

```

```

        friendpost_feed =
friendpost_feed.prefetch_related("jobspheres")
        friendpost_feed = friendpost_feed.filter(mappoint =
city_for_feed, toarchive = False)
        friendpost_feed = friendpost_feed.exclude(minuses__in =
[friendapplicant])
        friendpost_feed = friendpost_feed.order_by("-postdate")
        for friendpost in friendpost_feed:
            friendpost.photos =
Friendpostphoto.objects.filter(friendpost = friendpost)
            friendpost.graduations =
Friendpostgraduation.objects.select_related("jobsphere", "zvo")
            friendpost.graduations =
friendpost.graduations.filter(friendpost = friendpost)
            friendpost.answers_number =
Friendpostanswer.objects.filter(friendpost = friendpost).count()
            friendregeest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost.friendapplicant) |
Q(friendapplicant = friendpost.friendapplicant, requester =
friendapplicant))
            if friendregeest_result:
                friendpost.friendapplicant_friend =
friendregeest_result.requestresult
            else:
                friendpost.friendapplicant_friend = -1
        return render(request, "citymover/qfeed.html",
friendpost_feed)

def friendpost_detail_view(request, friendpostid):
    try:
        friendpost =
Friendpost.objects.prefetch_related("minuses")
        friendpost = friendpost.prefetch_related("hobbies")
        friendpost =
friendpost.prefetch_related("jobprofessions")
        friendpost =
friendpost.prefetch_related("jobspheres")
        friendpost = friendpost.select_related("mappoint")
        friendpost =
friendpost.select_related("friendapplicant").get(id =
friendpostid)
    except Friendpost.DoesNotExist:
        return render(request, "citymover/error.html",

```

```

        {"error": "Таке запитання щодо переїзду не
існує. Можливо неправильно заданий ідентифікатор."})
        if request.user.is_active:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").filter(appuser = request.user)
            friendpost.photos =
Friendpostphoto.objects.filter(friendpost = friendpost)
            friendpost.graduations =
Friendpostgraduation.objects.select_related("jobsphere",
"zvo").filter(friendpost = friendpost)
            friendpost.answers_number =
Friendpostanswer.objects.filter(friendpost = friendpost).count()
            friendpost_answers =
Friendpostanswer.objects.prefetch_related("pluses").filter(frien
dpost = friendpost)
            friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost.friendapplicant) |
Q(friendapplicant = friendpost.friendapplicant, requester =
friendapplicant))
            if friendrequest_result:
                friendpost.friendapplicant_friend =
friendrequest_result.requestresult
            else:
                friendpost.friendapplicant_friend = -1
                for friendpost_answer in friendpost_answers:
                    friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost_answer.friendapplicant)
| Q(friendapplicant = friendpost_answer.friendapplicant,
requester = friendapplicant))
                    if friendrequest_result:
                        friendpost_answer.friendapplicant_friend =
friendrequest_result.requestresult
                    else:
                        friendpost_answer.friendapplicant_friend = -1
                return render(request, "citymover/detailpost.html",
friendpost)

    def applicant_friendposts_view(request,
friendapplicant_name = None):
        if not friendapplicant_name:
            if request.user.is_active:

```

```

        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
        except Friendapplicant.DoesNotExist:
            return render(request,
"citymover/error.html", {"error": "Ви не внесли дані про себе в
особову картку, тому не можна сформувати перелік ваших запитань
щодо переїзду."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна
сформувати перелік ваших запитань щодо переїзду. Авторизуйтеся,
будь ласка, і поверніться до потоку."})
        else:
            try:
                friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser__username = friendapplicant_name)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
{"error": "Картки такого користувача немає в програмі.
Неправильно заданий логін."})
            friendpost_feed =
Friendpost.objects.prefetch_related("hobbies").prefetch_related(
"jobprofessions").prefetch_related("jobspheres")
            friendpost_feed =
friendpost_feed.filter(friendapplicant = friendapplicant)
            friendpost_feed = friendpost_feed.order_by("-postdate")
            for friendpost in friendpost_feed:
                friendpost.photos =
Friendpostphoto.objects.filter(friendpost = friendpost)
                friendpost.graduations =
Friendpostgraduation.objects.select_related("jobsphere",
"zvo").filter(friendpost = friendpost)
                friendpost.answers_number =
Friendpostanswer.objects.filter(friendpost = friendpost).count()
            return render(request, "citymover/ownposts.html",
friendpost_feed)

def applicant_answers_view(request, friendapplicant_name =
None):
    if not friendapplicant_name:
        if request.user.is_active:

```

```

        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
        except Friendapplicant.DoesNotExist:
            return render(request,
"citymover/error.html", {"error": "Ви не внесли дані про себе в
особову картку, тому не можна сформувати перелік ваших
відповідей на запитання щодо переїзду."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна
сформувати перелік ваших відповідей на запитання щодо переїзду.
Авторизуйтеся, будь ласка, і поверніться до потоку."})
        else:
            try:
                friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser__username = friendapplicant_name)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
{"error": "Картки такого користувача немає в програмі.
Неправильно заданий логін."})
            postanswer_feed =
Friendpostanswer.objects.select_related("friendapplicant")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__hobbies")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__jobprofessions")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__jobspheres")
            postanswer_feed =
postanswer_feed.prefetch_related("pluses")
            postanswer_feed =
postanswer_feed.filter(friendapplicant = friendapplicant)
            postanswer_feed = postanswer_feed.order_by("-
answerdate")
            for postanswer in postanswer_feed:
                postanswer.photos =
Friendpostanswerphoto.objects.filter(friendpostanswer =
postanswer)
            postanswer_answers =
Friendpostanswer.objects.select_related("friendapplicant")

```

```

        postanswer_answers =
postanswer_answers.prefetch_related("friendpost__hobbies")
        postanswer_answers =
postanswer_answers.prefetch_related("friendpost__jobprofessions"
)
        postanswer_answers =
postanswer_answers.prefetch_related("friendpost__jobspheres")
        postanswer_answers =
postanswer_answers.prefetch_related("pluses")
        postanswer_answers =
postanswer_answers.filter(friendpost = postanswer.friendpost)
        postanswer_answers =
postanswer_answers.order_by("answerdate")
        for postanswer_answer in postanswer_answers:
            friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost_answer.friendapplicant)
| Q(friendapplicant = friendpost_answer.friendapplicant,
requester = friendapplicant))
            if friendrequest_result:
                friendpost_answer.friendapplicant_friend =
friendrequest_result.requestresult
            else:
                friendpost_answer.friendapplicant_friend =
-1

        postanswer.answers = postanswer_answers
        friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester =
postanswer.friendpost.friendapplicant) | Q(friendapplicant =
postanswer.friendpost.friendapplicant, requester =
friendapplicant))
        if friendrequest_result:
            postanswer.friendapplicant_friend =
friendrequest_result.requestresult
        else:
            postanswer.friendapplicant_friend = -1
        return render(request, "citymover/ownanswers.html",
postanswer_feed)

def applicant_useful_answers_view(request):
    if request.user.is_active:
        try:

```

```

        friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
        except Friendapplicant.DoesNotExist:
            return render(request, "citymover/error.html",
{"error": "Ви не внесли дані про себе в особову картку, тому не
можна сформувати перелік корисних для вас відповідей на
запитання щодо переїзду."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна сформувати
перелік корисних для вас відповідей на запитання щодо переїзду.
Авторизуйтеся, будь ласка, і поверніться до потоку."})
            postanswer_feed =
Friendpostanswer.objects.select_related("friendapplicant")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__hobbies")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__jobprofessions")
            postanswer_feed =
postanswer_feed.prefetch_related("friendpost__jobspheres")
            postanswer_feed =
postanswer_feed.prefetch_related("pluses")
            postanswer_feed = postanswer_feed.filter(pluses__in =
[friendapplicant])
            postanswer_feed = postanswer_feed.order_by("-
answerdate")
            for postanswer in postanswer_feed:
                postanswer.photos =
Friendpostanswerphoto.objects.filter(friendpostanswer =
postanswer)
                postanswer_answers =
Friendpostanswer.objects.select_related("friendapplicant")
                postanswer_answers =
postanswer_answers.prefetch_related("friendpost__hobbies")
                postanswer_answers =
postanswer_answers.prefetch_related("friendpost__jobprofessions"
)
                postanswer_answers =
postanswer_answers.prefetch_related("friendpost__jobspheres")
                postanswer_answers =
postanswer_answers.prefetch_related("pluses")
                postanswer_answers =
postanswer_answers.filter(friendpost = postanswer.friendpost)

```

```

        postanswer_answers =
postanswer_answers.order_by("answerdate")
        for postanswer_answer in postanswer_answers:
            friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost_answer.friendapplicant)
| Q(friendapplicant = friendpost_answer.friendapplicant,
requester = friendapplicant))
            if friendrequest_result:
                friendpost_answer.friendapplicant_friend =
friendrequest_result.requestresult
            else:
                friendpost_answer.friendapplicant_friend =
-1

        postanswer.answers = postanswer_answers
        friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester =
postanswer.friendpost.friendapplicant) | Q(friendapplicant =
postanswer.friendpost.friendapplicant, requester =
friendapplicant))
        if friendrequest_result:
            postanswer.friendapplicant_friend =
friendrequest_result.requestresult
        else:
            postanswer.friendapplicant_friend = -1
        return render(request, "citymover/usefulanswers.html",
postanswer_feed)

def applicant_friends_view(request, friendapplicant_name =
None):
    if not friendapplicant_name:
        if request.user.is_active:
            try:
                friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
            except Friendapplicant.DoesNotExist:
                return render(request,
"citymover/error.html", {"error": "Ви не внесли дані про себе в
особову картку, тому не можна сформувати перелік ваших друзів у
додатку."})
        else:
            return render(request, "citymover/error.html",

```

```

        {"error": "Без авторизації не можна
сформувати перелік ваших друзів у додатку. Авторизуйтеся, будь
ласка, і поверніться до сторінки."})
    else:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser__username = friendapplicant_name)
        except Friendapplicant.DoesNotExist:
            return render(request, "citymover/error.html",
{"error": "Картки такого користувача немає в програмі.
Неправильно заданий логін."})
            applicant_friends =
Friendrequest.objects.select_related("friendapplicant")
            applicant_friends =
applicant_friends.select_related("requester")
            applicant_friends =
applicant_friends.filter(Q(friendapplicant = friendapplicant) |
Q(requester = friendapplicant), requestresult = 1)
            applicant_friends = applicant_friends.order_by("-
requestdate")
            return render(request, "citymover/friends.html",
applicant_friends)

def applicant_requests_view(request, addressed = True):
    if request.user.is_active:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
        except Friendapplicant.DoesNotExist:
            return render(request, "citymover/error.html",
{"error": "Ви не внесли дані про себе в особову картку, тому не
можна сформувати перелік запитів на дружбу в додатку."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна сформувати
перелік запитів на дружбу в додатку. Авторизуйтеся, будь ласка,
і поверніться до сторінки."})
            applicant_friends =
Friendrequest.objects.select_related("friendapplicant")
            applicant_friends =
applicant_friends.select_related("requester")
            if addressed:

```

```

        applicant_friends =
applicant_friends.filter(friendapplicant = friendapplicant,
requestresult = 0)
        else:
            applicant_friends =
applicant_friends.filter(requester = friendapplicant,
requestresult = 0)
            applicant_friends = applicant_friends.order_by("-
requestdate")
            return render(request, "citymover/friendrequests.html",
applicant_friends)

from django.http import HttpResponseRedirect
from .forms import FriendmessageForm

def applicant_friend_messages_view(request,
friendapplicant_name):
    if request.user.is_active:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
{"error": "Ви не внесли дані про себе в особову картку, тому не
можна переглядати напряму відправлені повідомлення."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна сформувати
перелік напряму відправлених повідомлень. Авторизуйтеся, будь
ласка, і поверніться до напряму відправлених повідомлень."})
        try:
            friendapplicant_addressant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser__username = friendapplicant_name)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
{"error": "Картки такого користувача немає в програмі.
Неправильно заданий логін."})
            communication_friend_messages =
Friendmessage.objects.select_related("friendrequest")
            communication_friend_messages =
communication_friend_messages.select_related("friendrequest__fri
endapplicant")

```

```

        communication_friend_messages =
communication_friend_messages.select_related("friendrequest__friendapplicant__appuser")
        communication_friend_messages =
communication_friend_messages.select_related("friendrequest__requester__appuser")
        communication_friend_messages =
communication_friend_messages.select_related("friendrequest__requester__appuser")
        communication_friend_messages =
communication_friend_messages.filter(Q(friendrequest__friendapplicant = friendapplicant, requester = friendapplicant_addressant)
| Q(friendrequest__friendapplicant = friendapplicant_addressant, requester = friendapplicant), friendrequest__requestresult = 1)
        communication_friend_messages =
communication_friend_messages.order_by("-friendmessagedate")
        if request.method == "POST":
            friendmessage_form =
FriendmessageForm(request.POST)
            if friendmessage_form.is_valid():
                base_request =
Friendrequest.objects.filter(friendapplicant = friendapplicant, requester = friendapplicant_addressant, requestresult = 1)
                base_request2 =
Friendrequest.objects.filter(friendapplicant =
friendapplicant_addressant, requester = friendapplicant, requestresult = 1)
                if base_request or base_request2:
                    friendmessage =
friendmessage_form.save(commit=False)
                    if base_request:
                        friendmessage.friendrequest =
base_request[0]
                        friendmessage.authorside = True
                    else:
                        friendmessage.friendrequest =
base_request2[0]
                        friendmessage.authorside = False
                    friendmessage.save()
                    return
            HttpResponseRedirect(reverse("citymover:applicantfriendmessagesview", args=(friendapplicant_name,)))
        else:
            friendmessage_form = FriendmessageForm()

```

```

        friendmessagedata = {}
        friendmessagedata["messages"] =
communication_friend_messages
        friendmessagedata["form"] = friendmessage_form
        return render(request, "citymover/messages.html",
friendmessagedata)

def friendapplicant_marks_view(request):
    if request.user.is_active:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
        except Friendapplicant.DoesNotExist:
            return render(request, "citymover/error.html",
{"error": "Ви не внесли дані про себе в особову картку, тому не
можна сформувати перелік відкладених вами карток."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна сформувати
перелік відкладених вами карток. Авторизуйтеся, будь ласка, і
поверніться до сторінки."})
            friendapplicantmarks =
Friendapplicantmark.objects.select_related("friendapplicant")
            friendapplicantmarks =
Friendapplicantmark.objects.select_related("friendapplicant__app
user")
            friendapplicantmarks =
Friendapplicantmark.objects.select_related("friendapplicant__map
point")
            friendapplicantmarks =
friendapplicantmarks.select_related("friendsaver")
            friendapplicantmarks =
Friendapplicantmark.objects.select_related("friendsaver__appuser
")
            friendapplicantmarks =
Friendapplicantmark.objects.select_related("friendsaver__mappoin
t")
            friendapplicantmarks =
friendapplicantmarks.filter(friendsaver = friendapplicant)
            friendapplicantmarks = friendapplicantmarks.order_by("-
markdate")
            for friendapplicantmark in friendapplicantmarks:

```

```

        friendrequest_result =
Friendrequest.objects.filter(friendapplicant =
friendapplicantmark.friendapplicant, requester =
friendapplicant)
        if friendrequest_result:
            friendapplicantmark.friendapplicant_friend =
friendrequest_result.requestresult
        else:
            friendapplicantmark.friendapplicant_friend = -1
        return render(request, "citymover/marks.html",
friendapplicantmarks)

def friendapplicant_card_view(request, friendapplicant_name
= None):
    login_friendapplicant = None
    if not friendapplicant_name:
        if request.user.is_active:
            try:
                login_friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser = request.user)
                friendapplicant = login_friendapplicant
            except Friendapplicant.DoesNotExist:
                return render(request,
"citymover/error.html", {"error": "Ви не внесли дані про себе в
особову картку, тому не можна її відобразити."})
        else:
            return render(request, "citymover/error.html",
{"error": "Без авторизації не можна
звернутися до цієї функції у такому варіанті. Авторизуйтеся,
будь ласка, і поверніться."})
    else:
        try:
            friendapplicant =
Friendapplicant.objects.select_related("mappoint",
"appuser").get(appuser__username = friendapplicant_name)
            except Friendapplicant.DoesNotExist:
                return render(request, "citymover/error.html",
{"error": "Картки такого користувача немає в програмі.
Неправильно заданий логін."})
            friendapplicant.photos =
Friendapplicantphoto.objects.filter(friendapplicant =
friendapplicant).order_by("-addeddate")

```

```

        jobstory =
Jobhistory.objects.select_related("jobprofession")
        jobstory =
jobstory.select_related("publicenterprisesphere")
        jobstory =
jobstory.select_related("publicenterprisesphere__publicenterpris
e")
        jobstory =
jobstory.select_related("publicenterprisesphere__publicenterpris
e__officemappoint")
        jobstory =
jobstory.select_related("publicenterprisesphere__jobsphere")
        friendapplicant.jobstories =
jobstory.filter(friendapplicant = friendapplicant)
        hobby =
Friendapplicanthobby.objects.select_related("hobby")
        friendapplicant.hobbies = hobby.filter(friendapplicant
= friendapplicant)
        graduation = Graduation.objects.select_related("zvo")
        graduation = graduation.select_related("jobsphere")
        graduation = graduation.filter(friendapplicant =
friendapplicant)
        friendapplicant.graduations = graduation.order_by("-
experienceend")
        languages =
Friendapplicantlanguage.objects.select_related("communicationlan
guage")
        languages = languages.filter(friendapplicant =
friendapplicant)
        friendapplicant.languages = languages
        if login_friendapplicant:
            ifmarked =
Friendapplicantmark.objects.filter(friendsaver =
login_friendapplicant, friendapplicant = friendapplicant)
            if ifmarked:
                friendapplicant.ifmarked = True
            else:
                friendapplicant.ifmarked = False
            ifrequested =
Friendrequest.objects.filter(Q(friendapplicant =
login_friendapplicant, requester =
friendapplicant))|Q(friendapplicant = friendapplicant, requester
= login_friendapplicant))
            if ifrequested:

```

```

        ifrequested = ifrequested[0]
        friendapplicant.ifrequested =
ifrequested.requestresult
        friendposts =
Friendpost.objects.select_related(mappoint)
        if not login_friendapplicant:
            friendposts = friendposts.filter(toarchive = False)
            friendposts = friendposts.filter(friendapplicant =
friendapplicant)
            friendposts = friendposts.prefetch_related("minuses")
            friendposts = friendposts.prefetch_related("hobbies")
            friendposts =
friendposts.prefetch_related("jobprofessions")
            friendposts =
friendposts.prefetch_related("jobspheres")
            friendposts = friendposts.order_by("-postdate")
            for friendpost in friendposts:
                friendpost.photos =
Friendpostphoto.objects.filter(friendpost = friendpost)
                graduations =
Friendpostgraduation.objects.select_related("jobsphere")
                graduations = graduations.select_related("zvo")
                graduations =
graduations.select_related("zvo__officemappoint")
                graduations = graduations.filter(friendpost =
friendpost)
                friendpost.graduations = graduations
                friendapplicant.posts = friendposts
                friendanswers =
Friendpostanswer.objects.select_related("friendpost")
                friendanswers =
Friendpostanswer.objects.prefetch_related("pluses")
                friendanswers = friendanswers.filter(friendapplicant =
friendapplicant)
                friendanswers = friendanswers.order_by("-answerdate")
                for friendanswer in friendanswers:
                    friendanswer.photos =
Friendpostanswerphoto.objects.filter(friendpostanswer =
friendanswer)
                return render(request, "citymover/applicant.html",
friendapplicant)

def friendpost_search_view(request):
    filter_mappoint = self.request.GET.get('point', None)

```



```

        filter_end = datetime.datetime(filter_endyear,
filter_endmonth, filter_endday)
        friendposts_found =
friendposts_found.filter(postdate__lte = filter_end)
        hobby = self.request.GET.get('hobby', None)
        if hobby:
            hobby = int(hobby)
            filter_hobby = Hobby.objects.get(id = hobby)
            friendposts_found =
friendposts_found.filter(hobbies__in = [filter_hobby])
        jobprofession = self.request.GET.get('prof', None)
        if jobprofession:
            jobprofession = int(jobprofession)
            filter_jobprofession = Jobprofession.objects.get(id
= jobprofession)
            friendposts_found =
friendposts_found.filter(jobprofessions__in =
[filter_jobprofession])
        jobsphere = self.request.GET.get('sphere', None)
        if jobsphere:
            jobsphere = int(jobsphere)
            filter_jobsphere = Jobsphere.objects.get(id =
jobsphere)
            friendposts_found =
friendposts_found.filter(jobspheres__in = [filter_jobsphere])
        zvo = self.request.GET.get('zvo', None)
        if zvo:
            zvo = int(zvo)
            filter_zvo = Zvo.objects.get(id = zvo)
            friendposts_found = friendposts_found.order_by("-
postdate")
            friendposts_found_all = []
            for friendpost in friendposts_found:
                graduation = None
                if zvo:
                    graduation =
Friendpostgraduation.objects.filter(friendpost = friendpost, zvo
= zvo)

                if (zvo and graduation) or (not zvo):
                    friendpost.photos =
Friendpostphoto.objects.filter(friendpost = friendpost)
                    friendpost.graduations =
Friendpostgraduation.objects.select_related("jobsphere", "zvo")

```

```

        friendpost.graduations =
friendpost.graduations.filter(friendpost = friendpost)
        friendpost.answers_number =
Friendpostanswer.objects.filter(friendpost = friendpost).count()
        friendrequest_result =
Friendrequest.objects.filter(Q(friendapplicant =
friendapplicant, requester = friendpost.friendapplicant) |
Q(friendapplicant = friendpost.friendapplicant, requester =
friendapplicant))
        if friendrequest_result:
            friendpost.friendapplicant_friend =
friendrequest_result.requestresult
        else:
            friendpost.friendapplicant_friend = -1
        friendpost.graduations =
Friendpostgraduation.objects.filter(friendpost = friendpost)
        friendposts_found_all.append(friendpost)
        hobbies = Hobby.objects.all().order_by("mainname")
        friendpostdata["hobbies"] = hobbies
        zvos = Zvo.objects.all().order_by("officialname")
        friendpostdata["zvos"] = zvos
        jobspheres =
Jobsphere.objects.all().order_by("spherename")
        friendpostdata["jobspheres"] = jobspheres
        jobprofessions =
Jobprofession.objects.all().order_by("mainname")
        friendpostdata["jobprofessions"] = jobprofessions
        friendpostdata["posts"] = friendposts_found_all
        return render(request, "citymover/search.html",
friendpostdata)

```

ДОДАТОК В
Слайди презентації

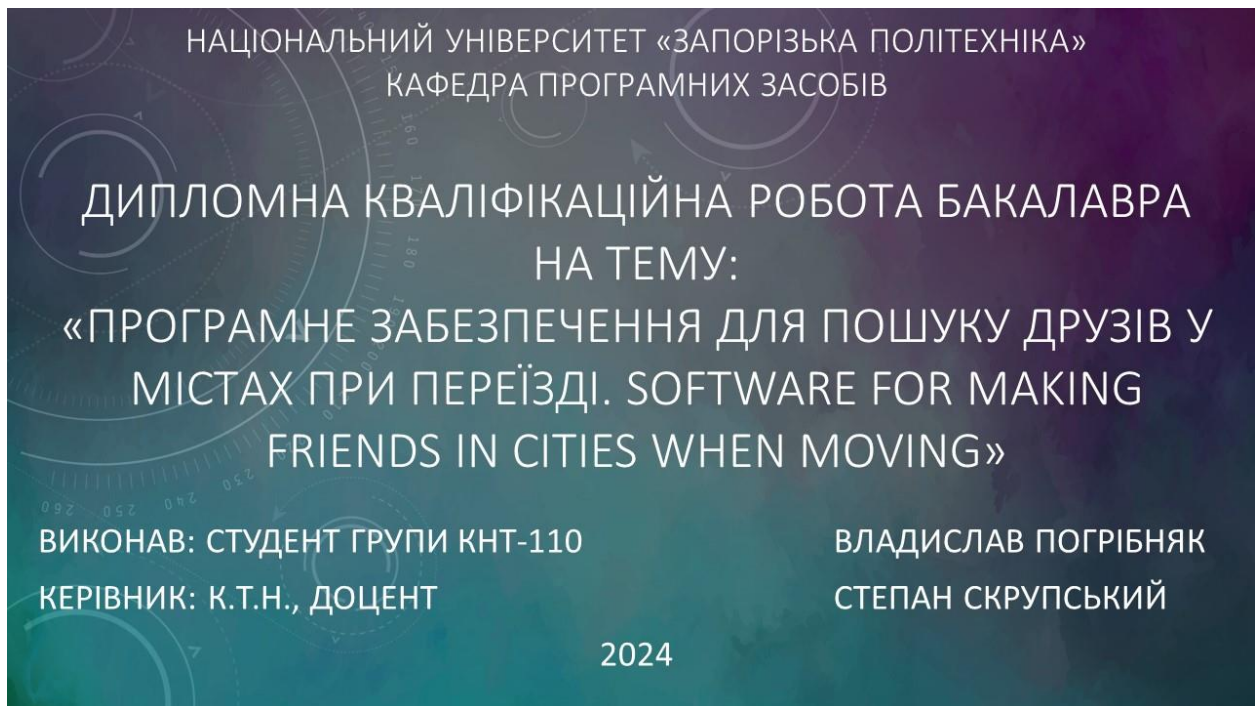


Рисунок В.1 – Початковий слайд

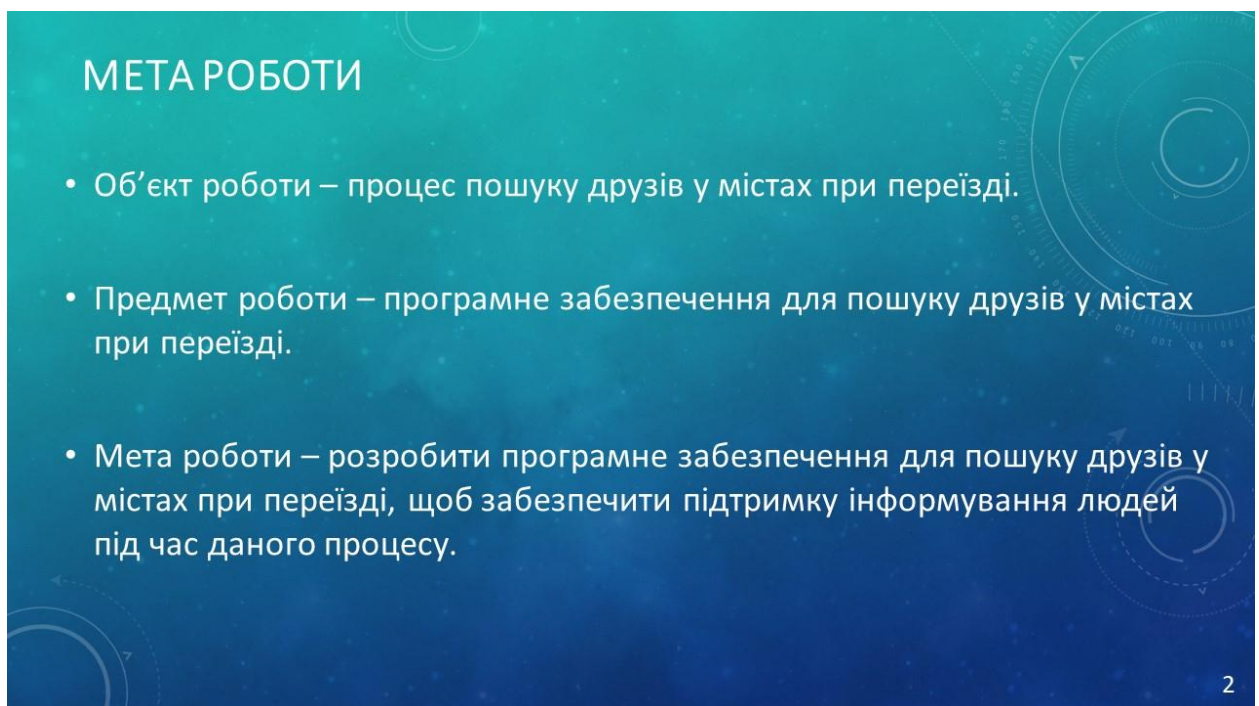


Рисунок В.2 – Мета роботи

ЗАВДАННЯ РОБОТИ

- проаналізувати програми для пошуку друзів у містах при переїзді;
- визначити засоби проектування програми;
- виділити вимоги до програми пошуку друзів у містах при переїзді;
- виконати реалізацію програми пошуку друзів у містах при переїзді.

3

Рисунок В.3 – Завдання роботи

ПОРІВНЯННЯ ЗАСОБІВ АНАЛОГІВ ЩОДО ПОШУКУ ДРУЗІВ У МІСТАХ ПРИ ПЕРЕЇЗДІ

Критерій	LinkedIn	Facebook	CITYMover
Пошук запитань щодо переїзду в інше місто	–	В межах спільноти серед всіх публікацій	+
Наявність окремого потоку запитань щодо переїзду в інше місто	–	–	+
Встановлення контактів, у тому числі в обмежених спільнотах	+	+	+
Деталізоване і структуроване представлення власної освіти, досвіду роботи, хобі	+	–	+
Зв'язування запитань з професією, хобі	–	–	+

4

Рисунок В.4 – Порівняння засобів аналогів щодо пошуку друзів у містах при переїзді

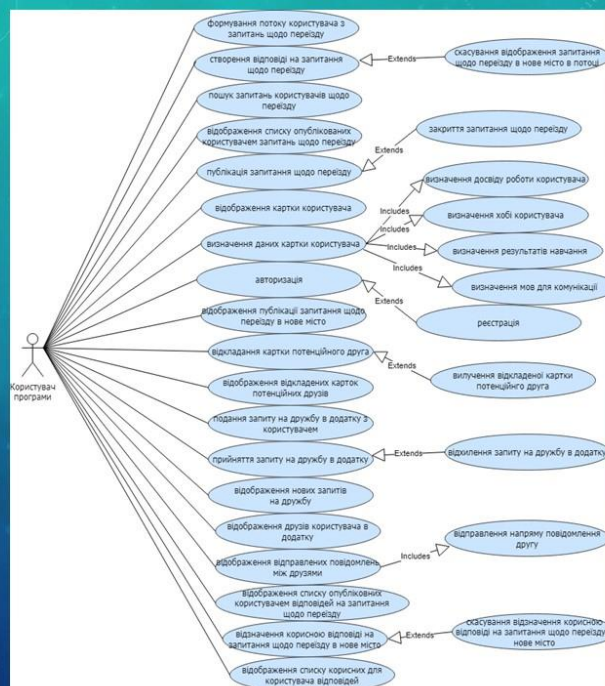
ПОРІВНЯННЯ ВЕБФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ ПРОГРАМИ ДЛЯ ПОШУКУ ДРУЗІВ У МІСТАХ ПРИ ПЕРЕЇЗДІ

Критерій	Laravel	Django
Мова програмування	PHP	Python, що має більшу простоту синтаксису, легкість вивчення тощо
Продуктивність	Нижча за рахунок великої кількості вбудованих засобів, що можуть бути непотрібними	Вища
Підтримувані бази даних	MySQL, PostgreSQL, SQLite	MySQL, Oracle, MariaDB, PostgreSQL, SQLite
Архітектура	Model-View-Controller	Model-View-Template
Рівень безпеки	Підвищується, але потребує пошуку та контролю	Високий за рахунок наявних інтегрованих засобів, що охоплюють величезну кількість проблем
Підтримка	Добра	Дещо краща за рахунок широкого кола документації

5

Рисунок В.5 – Порівняння вебфреймворків для розробки програми для пошуку друзів у містах при переїзді

ДІАГРАМА ПРЕЦЕДЕНТІВ ПРОГРАМИ ДЛЯ ПОШУКУ ДРУЗІВ У МІСТАХ ПРИ ПЕРЕЇЗДІ



6

Рисунок В.6 – Діаграма прецедентів програми для пошуку друзів у містах при переїзді

СХЕМА ФУНКЦІОНУВАННЯ ПРОГРАМИ ДЛЯ ПОШУКУ ДРУЗІВ У МІСТАХ ПРИ ПЕРЕЇЗДІ

```

graph TD
    Start([початок роботи]) --> Search[пошук запитань щодо переїзду в інше місто]
    Search --> Decision{доступна функція}
    Decision --> Auth[авторизація]
    Decision --> Stream[відображення потоку запитань щодо переїзду]
    Decision --> Search2[пошук запитань щодо переїзду в інше місто]
    Decision --> Pub[відображення опублікованих користувачем запитань]
    Decision --> NewPub[публікація нового запитання щодо переїзду в інше місто]
    Decision --> Answers[відображення корисних відповідей]
    Decision --> Cards[відображення відкладених карток]
    Decision --> FriendsList[відображення запитів на дружбу в додатку]
    Decision --> Friends[відображення друзів у додатку]
    Decision --> Profile[змінення власної картки]
    Decision --> End([завершення роботи])
  
```

The flowchart illustrates the functional scheme of a program for finding friends in cities during a move. It begins with 'початок роботи' (start of work), leading to 'пошук запитань щодо переїзду в інше місто' (search for questions about moving to another city). A decision diamond 'доступна функція' (function available) branches into ten parallel options: 'авторизація' (authorization), 'відображення потоку запитань щодо переїзду' (display of a stream of questions about moving), 'пошук запитань щодо переїзду в інше місто' (search for questions about moving to another city), 'відображення опублікованих користувачем запитань' (display of questions published by the user), 'публікація нового запитання щодо переїзду в інше місто' (publication of a new question about moving to another city), 'відображення корисних відповідей' (display of useful answers), 'відображення відкладених карток' (display of saved cards), 'відображення запитів на дружбу в додатку' (display of friendship requests in the app), 'відображення друзів у додатку' (display of friends in the app), and 'змінення власної картки' (changing your own card). All paths converge at 'завершення роботи' (end of work).

Рисунок В.8 – Схема функціонування програми для пошуку друзів у містах при переїзді

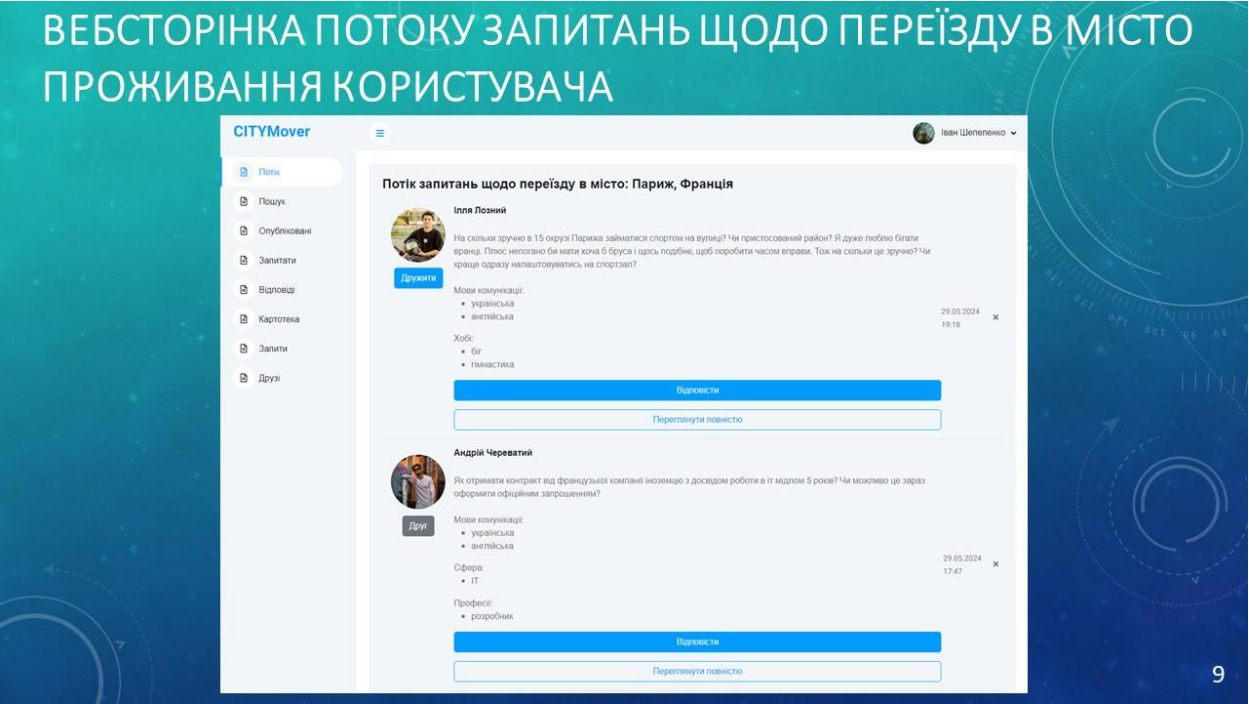


Рисунок В.9 – Вебсторінка потоку запитань щодо переїзду в місто проживання користувача

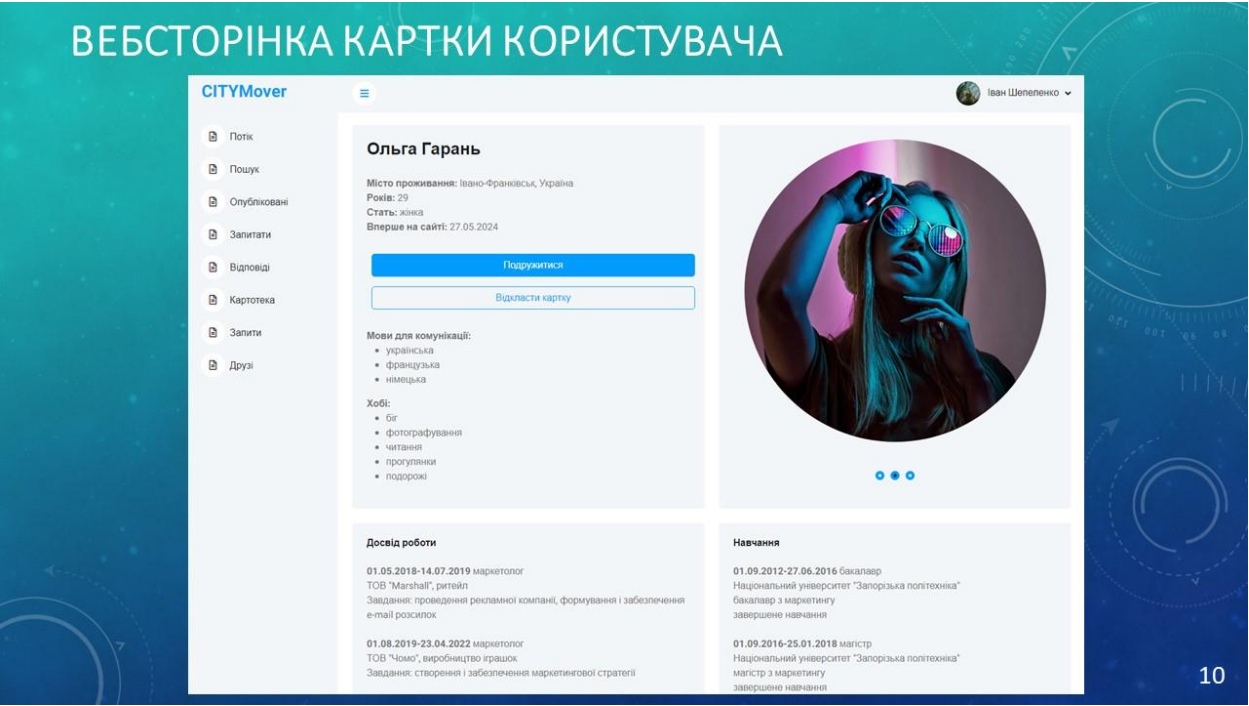


Рисунок В.10 – Вебсторінка картки користувача

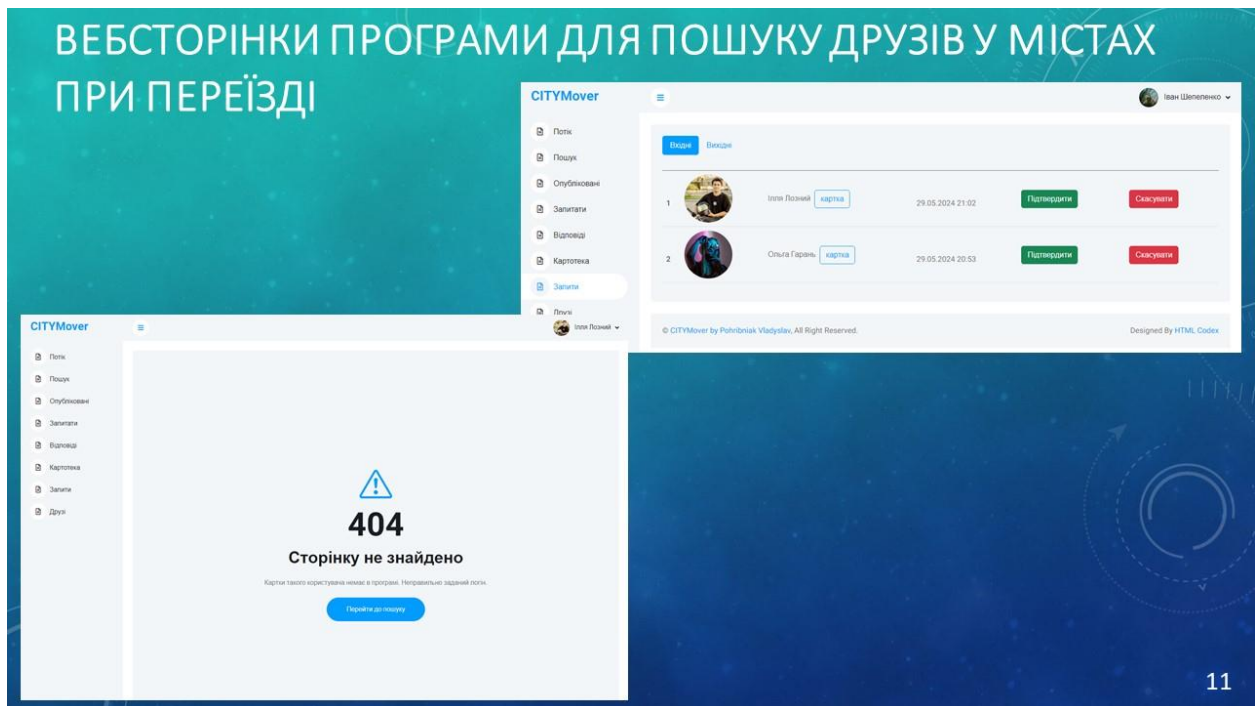


Рисунок В.11 – Вебсторінки програми для пошуку друзів у містах при переїзді

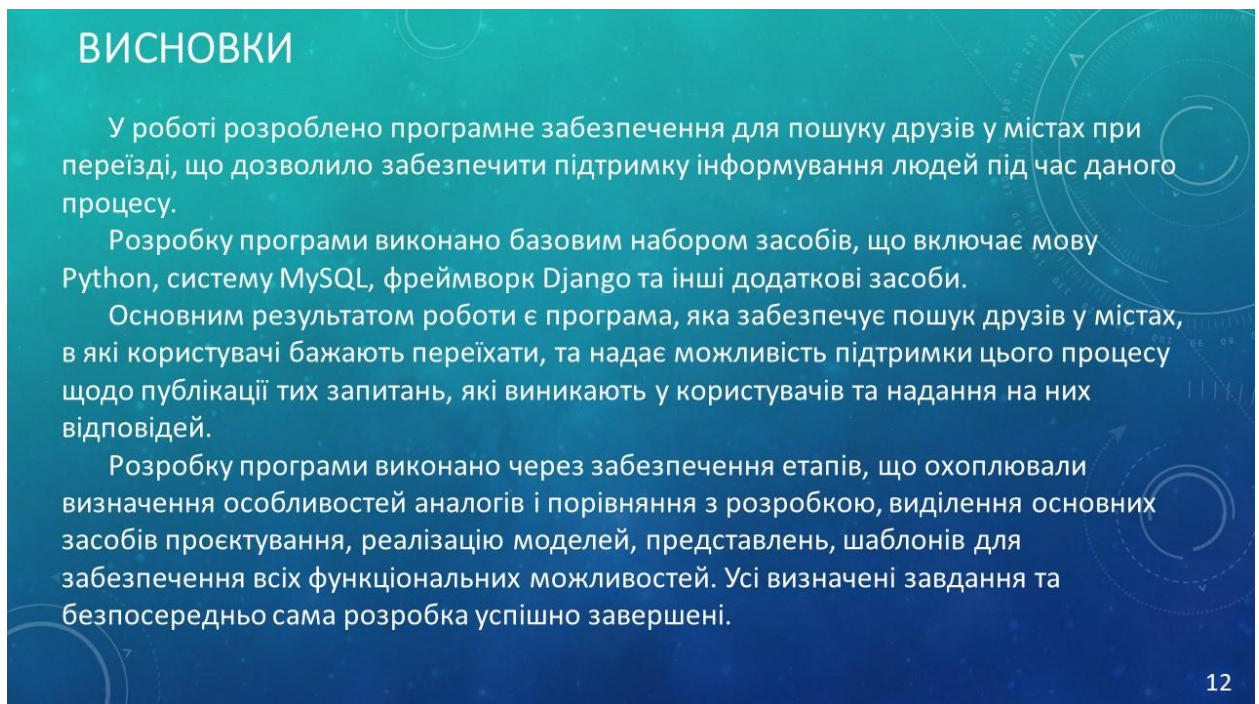


Рисунок В.12 – Висновки