НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам

Виконав студент IV курсу, групи ІТ-82
(шифр групи)
Лозяк Мар'яна Василівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Сирота О. П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІБ, к.т.н., доц., Родіонов А. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Лозяк Мар'яні Василівні
(прізвище, ім'я, по батькові)

1. Тема проєкту Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам

керівник проєкту Сирота Олена Петрівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 10 »червня 2022 р. №1033-с

2. Термін подання студентом проєкту « 19 » червня 2022 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: загальні положення, опис і аналіз предметної області, аналіз успішних ІТ-проєктів, аналіз вимог до програмного забезпечення, постановка комплексу завдань модуля.
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, архітектура програмного забезпечення.
3) Розробка та реалізація клієнтської частини: React, Redux, MaterialUI.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування.
5) Впровадження та супровід програмного забезпечення: розгортання програмного забезпечення, робота з програмним забезпеченням.

5. Перелік графічного матеріалу

1) Схема бізнес-процесу _____

2) Схема структурна послідовностей _____

3) Креслення вигляду екранних форм _____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2022 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2022	
2	Аналіз існуючих методів розв’язання задачі	28.04.2022	
3	Постановка та формалізація задачі	01.05.2022	
4	Розробка інформаційного забезпечення	03.05.2022	
5	Алгоритмізація задачі	07.05.2022	
6	Обґрунтування вибору використаних технічних засобів	10.05.2022	
7	Розробка програмного забезпечення	24.05.2022	
8	Налагодження програми	25.05.2022	
9	Виконання графічних документів	01.06.2022	
10	Оформлення пояснювальної записки	05.06.2022	
11	Подання ДП на попередній захист	06.06.2022	
12	Подання ДП рецензенту	12.06.2022	
13	Подання ДП на основний захист	19.06.2022	

Студент _____ Мар’яна Лозяк
(підпис) (ініціали, прізвище)

Керівник _____ Олена Сирота
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’ятьох розділів, містить 19 таблиць, 22 рисунки та 14 джерел – загалом 68 сторінок.

Дипломний проєкт присвячений конструюванню та розробці системи, що надасть можливість людям розміщувати пропозиції про можливість допомогти ВПО чи повідомлення про потреби.

Метою проєкту є спрощення процесу знаходження можливих способів допомоги внутрішньо переміщеним особам.

Об'єкт дослідження: система обміну пропозиціями про безкоштовну допомогу.

Предмет дослідження: процес обміну пропозиціями про безкоштовну допомогу, їх розміщення та поширення.

У першому розділі проведено аналіз предметної області та вимог до системи, сформовано функціональні та нефункціональні вимоги.

Другий розділ присвячено моделюванню та описанню архітектури. Описано розробку серверної частини застосунку з допомогою сервісів, що надає Firebase. Описано структуру бази даних та створено необхідні функції.

У третьому розділі описано процес проектування інтерфейсу користувача та структуру клієнтської частини застосунку.

У четвертому розділі було проведено аналіз якості застосунку та наведено опис тестування основних функціональних можливостей.

П’ятий розділ присвячено впровадженню створеного застосунку та описано покроковий процес розгортання.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, SERVERLESS, GOOGLE FIREBASE, БАЗА ДАНИХ, ПРОПОЗИЦІЇ ДОПОМОГИ, REACT.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 19 tables, 22 figures and 14 sources – in total 68 pages.

The diploma project is dedicated to the design and development of a system that will enable people to post suggestions about the possibility of helping IDPs or messages about needs.

The aim of the project is to simplify the process of finding possible ways to help internally displaced persons.

Object of research: information system for help proposal exchange.

Subject of research: the process of exchanging proposals for free assistance, their placement and distribution.

In the first section, the subject area and system requirements are analyzed, functional and non-functional requirements are formed.

The second section is devoted to the modeling and description of architecture. The development of the server part of the application using the services provided by Firebase is described. The database structure is defined and created.

The third section describes the process of designing the user interface and the structure of the client part of the application.

In the fourth section, the quality of the application is analyzed and a description of the testing of the main functionality is provided.

The fifth section is devoted to the implementation of the created application and describes the step-by-step deployment process.

KEYWORDS: WEB APPLICATION, SERVERLESS, GOOGLE FIREBASE, DATABASE, SUGGESTIONS FOR HELP, REACT.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-8216.045440.01.91	Технічне завдання	10	
3	А4	КПІ.ІТ-8216.045440.02.81	Пояснювальна записка	68	
4	А4	КПІ.ІТ-8216.045440.03.13	Опис програми	22	
5	А4	КПІ.ІТ-8216.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-8216.045440.05.34	Керівництво користувача	15	
7	А3	КПІ.ІТ-8216.045440.06.99.СБП	Схема бізнес-процесу	1	
8	А3	КПІ.ІТ-8216.045440.06.99.ССП	Схема структурна послідовностей	1	
9	А3	КПІ.ІТ-8216.045440.06.99.КЕ	Креслення вигляду екранних форм	1	
10	А4		Звіт Unicheck	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-8216.045440.00.90		
Розроб.		Лозяк М.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Сирота О.П.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-82	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ВЕБ-ЗАСТОСУНОК ОБМІНУ ПРОПОЗИЦІЯМИ ДЛЯ ДОПОМОГИ ВНУТРІШНЬО ПЕРЕМІЩЕНИМ ОСОБАМ

Технічне завдання

КПІ.ІТ-8216.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена СИРОТА

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Мар’яна ЛОЗЯК

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам.

Галузь застосування: Наведене технічне завдання поширюється на розробку веб-застосунку обміну пропозиціями для допомоги внутрішньо переміщеним особам [КПІ.ІТ-8216.045440], котра використовується для розміщення та пошуку пропозицій про безоплатну допомогу та призначена для використання внутрішньо переміщеними особами (чи іншими людьми, що потребують підтримки) та людьми, що бажають надати допомогу.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосунку обміну пропозиціями для допомоги внутрішньо переміщеним особам є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для розміщення повідомлень про можливі шляхи надання допомоги (та про потреби в допомозі).

Метою розробки є спрощення процесу знаходження можливих способів допомоги внутрішньо переміщеним особам (чи іншим особам, що потребують допомоги) шляхом надання можливості розміщувати оголошення, здійснювати пошук по них тощо.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції для користувача:

– перегляд розміщених оголошень;

– фільтрування оголошень (вибір між пропозиціями та потребами, фільтр по категорії тощо);

– пошук оголошень;

– додавання нового оголошення;

– видалення оголошення;

– реєстрація;

– авторизація.

4.1.2 Розробку виконати на платформі Windows 10.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування

Не висуваються.

4.3.3 Обслуговуючий персонал

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору Intel i3.

4.4.2.2 Підтримка Google Chrome 60+, Microsoft Edge 15+, Firefox 56+ або Safari 10+.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows 7 або вище, Unix.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: текстові та інші дані, отримані в результаті введення користувачем у відповідні поля та отримані від елементів керування, що формують HTTP-запити на сервер з тілом у форматі JSON чи form-data і методами GET, POST, DELETE.

4.5.3 Результати повинні бути представлені в наступному форматі: відповіді на HTTP-запити у вигляді JSON, елементи користувацького інтерфейсу програми у вікні браузера.

4.5.4 Програмне забезпечення повинно бути створеним на мові JavaScript з використанням node.js.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Розгорнути вебзастосунок.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Програма та методика тестування.

5.4 Графічна частина повинна бути виконана не менше ніж на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема бізнес-процесу.

5.4.2 Схема структурна послідовностей.

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	10.03.2022	
2.	Аналіз існуючих методів розв’язання задачі	28.04.2022	Технічне завдання
3.	Постановка та формалізація задачі	01.05.2022	Специфікації програмного забезпечення
4.	Розробка інформаційного забезпечення	03.05.2022	Схема бізнес процесу та діаграма послідовностей
5.	Алгоритмізація задачі	07.05.2022	Специфікація компонентів
6.	Обґрунтування вибору використаних технічних засобів	10.05.2022	Технічна документація
7.	Розробка програмного забезпечення	24.05.2022	Тексти програмного забезпечення
8.	Налагодження програми	25.05.2022	Тести, результати тестування
9.	Виконання графічних документів	01.06.2022	Графічний матеріал проекту
10.	Оформлення пояснювальної записки	05.06.2022	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам

КПІ.ІТ-8216.045440.02.81

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application programming interface, прикладний програмний Інтерфейс.

БД – База даних.

ВПО – Внутрішньо переміщена особа.

BaaS – Backend-as-a-Service, платформа, що автоматизує внутрішню розробку та на надає можливість пов’язати програми з хмарним сховищем та API.

FaaS – Function-as-a-Service, сервіс, що надає можливість виклику екземпляру керуючого коду без необхідності керування серверами.

REST – Representational State Transfer, підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

JSON – JavaScript Object Notation, текстовий формат обміну даними між комп'ютерами, базується на тексті, може бути прочитаним людиною.

NPM – Node Package Manager, менеджер пакунків для мови програмування JavaScript.

SSL – Secure Sockets Layer, криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером.

CDN – Content Distribution Network, географічно розподілена мережева інфраструктура, що дозволяє оптимізувати доправлення та розповсюдження контенту кінцевим користувачам в мережі Інтернет.

HTTPS – HyperText Transfer Protocol Secure, розширення протоколу HTTP для підтримки шифрування в цілях підвищення безпеки.

RSA – криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел.

ECDSA – Elliptic Curve Digital Signature Algorithm, алгоритм з відкритим ключем для створення цифрового підпису.

Serverless – модель хмарних обчислень для яких платформа динамічно керує виділенням машинних ресурсів.

ВСТУП

За результатами дослідження проведеного Міжнародною організацією з міграції станом на 3 травня 2022 число внутрішньо переміщених осіб в Україні становить понад 8 мільйонів людей.[1] Серед цієї великої групи ВПО, як і серед будь-якої іншої групи, є люди з різними рівнями достатку, різними рівнями соціальної захищеності. Та й ситуація, що склалась у кожного, хто змінив за час війни місце проживання, може сильно відрізнятись. Саме тому стверджувати, що всі 8 мільйонів потребують допомоги неможливо. Проте, кількість тих, кому вона й справді потрібна, досить значна.

Для підтримки ВПО майже в кожному місті створюють центри гуманітарної допомоги. Чимало громадян намагаються внести свою лепту в наданні помочі. І дуже часто повідомлення про допомогу розміщують на сайтах громад, на сторінках у соцмережах чи поширюють іншим чином. Проте у великому інформаційному вирі сьогодні є великий шанс втратити потрібний контакт чи пропустити пропозицію.

До розв'язання завдання пошуку людей, що можуть надати необхідну підтримку, долучається велика кількість громадян. Таким чином з’явились деякі наявні зараз програмні продукти для пошуку житла чи пошуку волонтерів у певній місцевості. У них пропозиції розміщуються у вигляді оголошень. Та створені рішення іноді надають лише можливість фільтрації пропозицій, іноді покривають лише частину вимог, іноді мають складну систему додавання оголошень.

Програмні рішення, у яких користувачу потрібно залишати заявки чи спеціальні запити для того, щоб отримати можливість підтримати когось, можуть викликати у людей сумніви у їх готовності зробити такий крок (наприклад, користувач може хвилюватись про зобов’язання, які накладаються на нього при поданні заявки тощо).

Саме тому ціллю створення даної системи є надання простого способу поєднати людей, що можуть допомогти, та людей, які цієї допомоги потребують. У користувачів має бути легкий спосіб переглянути доступні пропозиції або ж самостійно обрати, кому допомогти. Перешкоди для людей, що можливо мають

певні хвилювання щодо того чи й справді варто розмістити повідомлення і протягнути руку, повинні бути мінімальними.

Створену систему можна буде використати в майбутньому для допомоги не лише внутрішньо переміщеним особам, але й просто для особистої волонтерської діяльності. Ресурс також може виступати в ролі платформи для розміщення безкоштовних пропозицій.

АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Сьогодні у сторінках у соцмережах дуже часто можна побачити оголошення про необхідні речі для ВПО. Найбільш популярними є запити про фінансову допомогу, проте часто трапляються оголошення на кшталт: “У центр допомоги переселенцям потрібні літні речі, ковдри, дитячі коляски тощо”. В той самий час нерідко з’являються повідомлення про доступні пропозиції допомоги, до прикладу: “Можу допомогти дістатись до кордону”, “Проводяться безкоштовні заняття малювання для дітей ВПО”. Такі оголошення розміщують на сторінках у соцмережах, у місцевих групах тощо.

В різноманітності джерел інформації можна загубити чи не помітити потрібне оголошення, особливо, якщо пропозиція досить локальна і розміщена, наприклад, на вебсайті міста.

Швидкого надання допомоги потребує чимала кількість людей. Так, наприклад, лише у Києві станом на 13 травня 2022 року зареєструвалися вже понад 47 тис. внутрішньо переміщених осіб. Це число включає лише людей, що офіційно отримали статус ВПО, тому реальні цифри значно більші.

У зв’язку з виникненням такої потреби з початком війни в Україні почали з’являтись сервіси, що дозволяють допомагати людям. Так одною з найбільших потреб була і залишається надання житла. Для того, щоб спростити цей процес було створено проєкт (вебресурс) “Прихисток”, що має на меті допомогти ВПО у пошуках помешкання.

Завдання пошуку чи розміщення оголошень про пропозиції допомоги звичайно ж може бути вирішене (і вирішується) розміщенням повідомлень у соціальних мережах чи, наприклад, на одному з вже існуючих сервісів для оголошень. Проте у даній роботі фокус напрямлений саме на створення системи, що буде призначена лише для надання безоплатної допомоги у зручній формі.

1.2 Опис і аналіз предметної області

Предметною областю в даній роботі є обмін пропозиціями про безкоштовну допомогу, їх розміщення та поширення.

Відповідно до досліджень платформ великих даних VKURSI та Zagoriy Foundation[2] значно збільшилась частота реєстрації благодійних організацій в нашій країні. Так, з більш ніж 15 тисяч існуючих в Україні доброчинних організацій, 8% було створено у квітні 2022 року. Такі дані досить добре ілюструють бажання і готовність громадян допомагати. Оскільки складно проаналізувати скільки саме пропозицій чи повідомлень про можливу допомогу поширюється в українському сегменті мережі Інтернет щодня, то по наведеній вище статистиці можна зробити висновок, що українці мають бажання і змогу допомагати. Тому й кількість пропозицій є великою.

В той самий час, можна проаналізувати, наприклад, кількість запитів google за пошуковим терміном “гуманітарна допомога переселенцям” з допомогою сервісу Google Trends.

Так відповідно до графіка, що наведений нижче на рисунку 1.1, питання залишається актуальним.

Рисунок 1.1 - Популярність пошукового запиту “гуманітарна допомога переселенцям” в період лютий-травень 2022 р.

За результатами звіту про внутрішнє переміщення в Україні Міжнародної організації з міграції станом на 3 травня 2022 року кількість внутрішньо

переміщених осіб в Україні становить приблизно 8 мільйонів людей. Така велика кількість людей прямопропорційно збільшує кількість запитів про допомогу[1].

На рис. 1.2 відображено статистику потреб опитаних в різних куточках України ВПО. Видно, що хоч основною потребою на цей момент люди називають фінанси, проте інші, такі як їжа, одяг, транспорт та ліки, не втрачають актуальності.

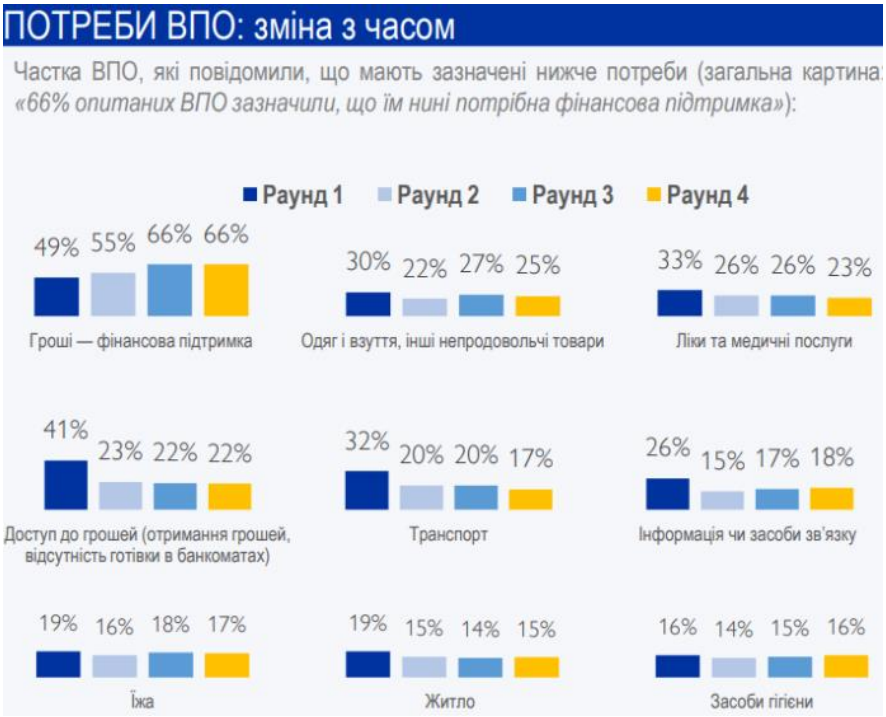


Рисунок 1.2 – Статистика зміни потреб ВПО з часом

Тому створення єдиного ресурсу для пошуку та розміщення оголошень є доцільним рішенням, для зручності обміном інформацією.

Електронна дошка оголошень дозволить розмістити інформацію про, наприклад, можливість надання певних послуг безкоштовно. Для цього повідомлення повинно містити дані про категорію, змістовну назву, населений пункт, в якому воно є актуальним. Ці дані дозволять полегшити сприйняття інформації.

1.3 Аналіз успішних IT-проєктів

Зараз вже існують деякі програмні проєкти, вебресурси, що дозволяють вирішити поставлені завдання. Деякі з них розв'язують проблему більш вузько (як, наприклад, пошук житла), деякі є новоствореними доповненнями до вже існуючих

платформ розміщення оголошень. В наступних підпунктах буде розглянуто деякі з них. Варто зазначити, що певні проєкти з’явились вже під час виконання даної роботи.

1.3.1 Аналіз відомих технічних рішень

Загалом можна розглянути кілька варіантів вже існуючих рішень. Розміщення оголошень для допомоги може бути здійснено на вебплатформах оголошень, при цьому, наприклад, вказуючи, що пропозиція безкоштовна. Іншим варіантом є користування ботом в одному з популярних месенджерів.

1.3.2 Аналіз відомих програмних продуктів

Один з найпопулярніших застосунків для допомоги - проєкт “Прихисток”. Він призначений допомогти у пошуку житла. Крім того, зараз цей ресурс є частиною державної програми повернення коштів, тому є досить популярним.

Нижче(рис. 1.3) наведено вигляд інтерфейсу застосунку.

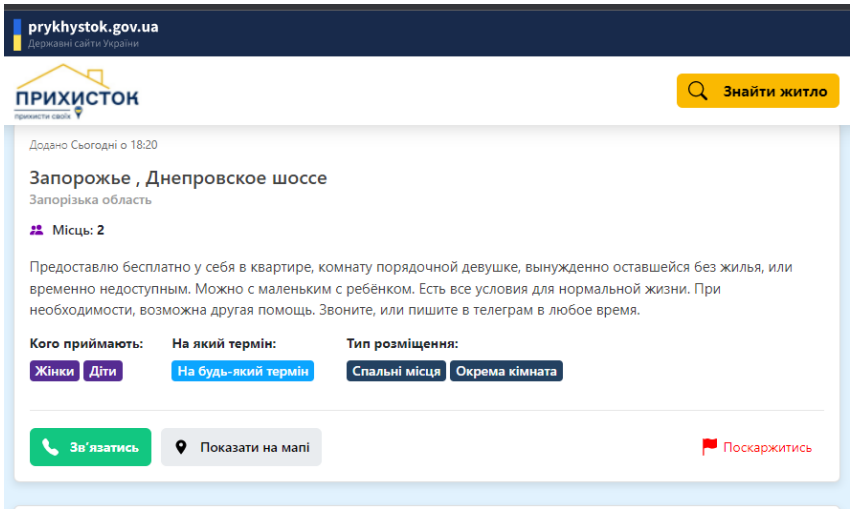


Рисунок 1.3 – Інтерфейс одної зі сторінок проєкту “Прихисток”

Переваги:

– зручний інтерфейс;
– можливість пошуку з фільтрами;
– відображення пропозицій на карті;
– легка навігація.

Складно виділити недоліки. Можна зазначити лише те, що сервіс дозволяє допомагати лише з пошуком житла, але це і є основною ідеєю цього проєкту.

Іншим уже існуючим успішним рішенням є Telegram-бот проєкту SaveUA. На скріншоті нижче наведено опис роботи бота.

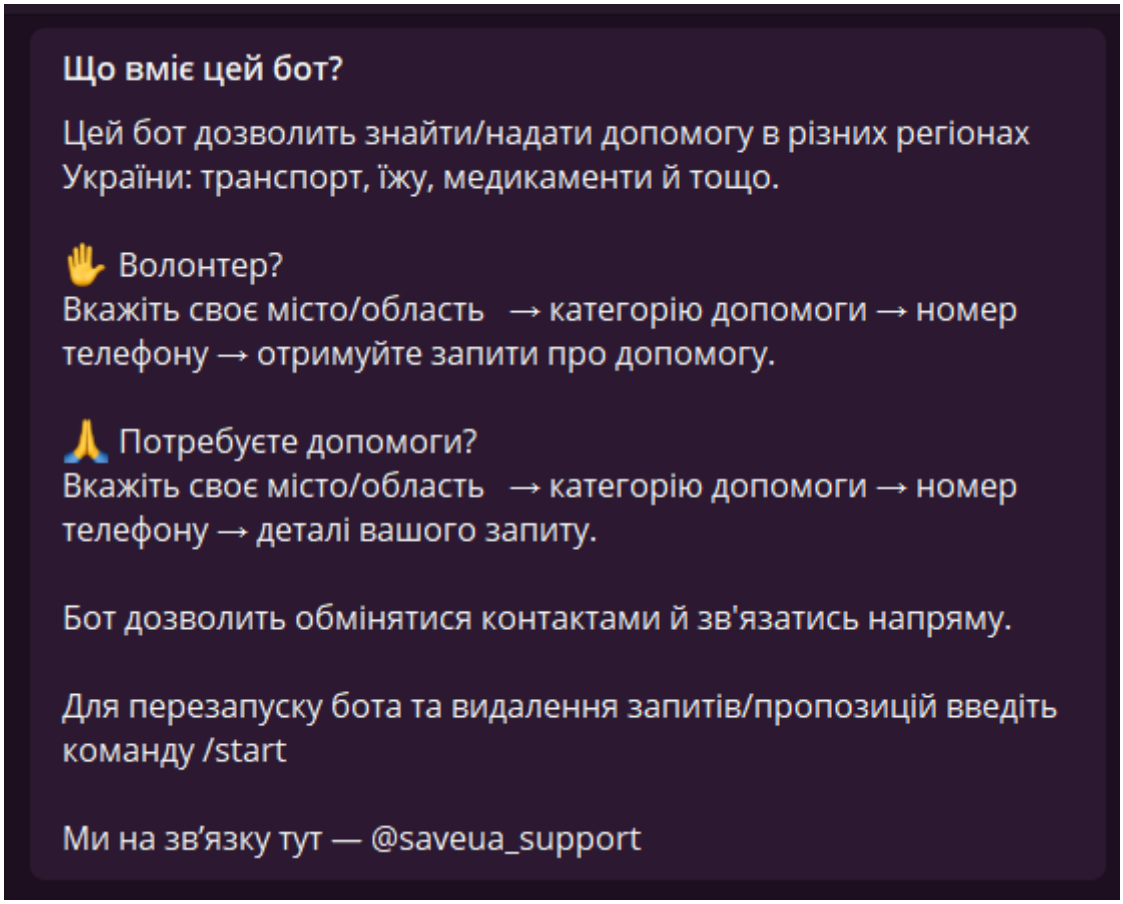


Рисунок 1.4 – Опис функціонала Telegram-бота SaveUA

Переваги:

– простота створення чи пошуку пропозиції допомоги;

– доступ через месенджер, який зазвичай вже встановлений у великої кількості користувачів;

– наявність категоризації.

Недоліки:

– обмеженість у специфікації опису пропозиції (можна вказати лише місто та категорію);

– необхідність у наявності даного месенджера.

Останнім у списку представлених програмних рішень в цьому розділі є найновішим. Це безкоштовна дошка оголошень “Potreba” представлена платформою для пошуку роботи robota.ua.

Інтерфейс даного рішення наведено на рис. 1.5.

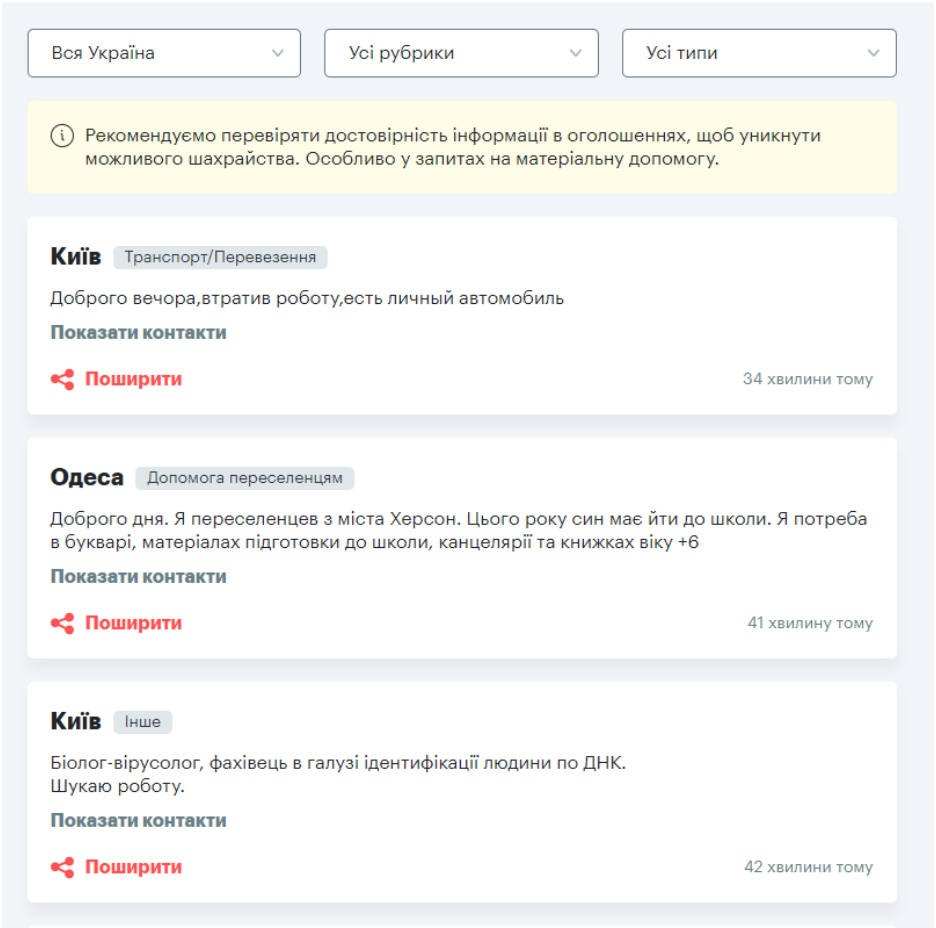


Рисунок 1.5 – Інтерфейс ресурсу “Potreba”

До переваг можна віднести простий та інтуїтивно зрозумілий інтерфейс, можливість додавання як оголошень про допомогу, так і повідомлень про потребу в допомозі.

Серед недоліків можна виділити те, що можна шукати оголошення лише за місцем та рубрикою.

Одним з найбільших існуючих зараз проєктів є створений Міністерством цифрової трансформації ресурс “єДопомога”.

До переваг цього програмного рішення можна віднести лаконічний та зручний в користуванні дизайн інтерфейсу, модерація оголошень

адміністраторами, що забезпечує достовірність інформації, вміст доступний кількома мовами, можливість надати швидку фінансову допомогу, якщо людина бажає допомогти, але при цьому не витрачати багато часу. Зареєструвавшись, користувач має можливість здійснювати фільтрацію оголошень.

Серед недоліків можна виділити те, що громадянин повинен спершу пройти реєстрацію для того, щоб переглянути потреби. Це може, наприклад, викликати у користувачів сумніви, щодо того чи будуть вони зобов'язані допомогти після реєстрації.

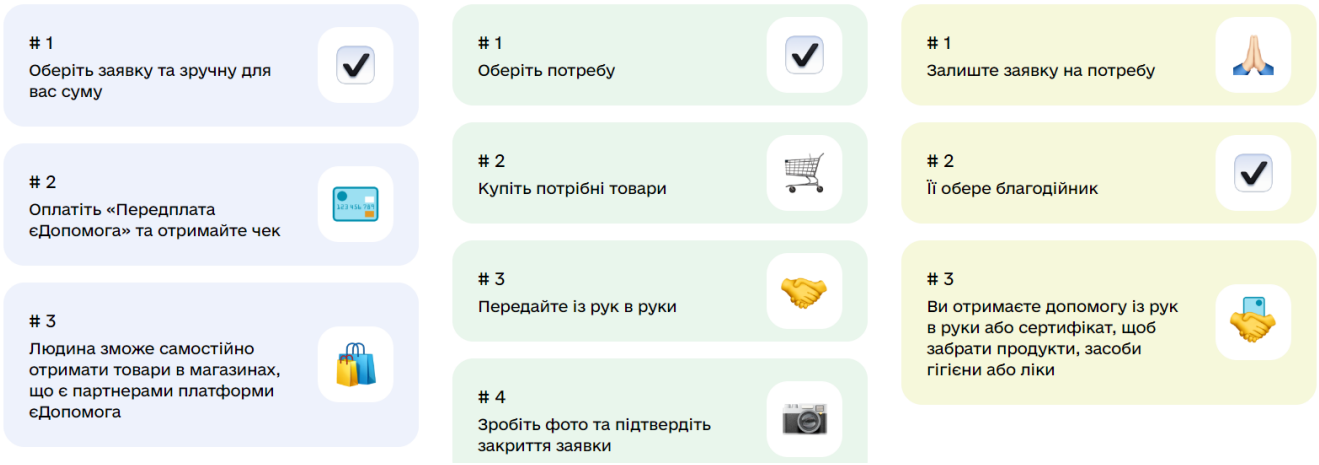


Рисунок 1.6 – Опис процедури користування сервісом "єДопомога"

1.4 Аналіз вимог до програмного забезпечення

Основна ціль створюваної системи полягає у можливості розміщення повідомлень про шляхи надання допомоги, або в розміщенні повідомлень про потребу в допомозі. Таким чином дана система надаватиме змогу "обмінюватись" пропозиціями допомоги.

1.4.1 Розроблення функціональних вимог

Оскільки в даній системі основними дійовими особами виступають лише волонтери (чи ті, хто просто мають змогу допомогти) та ВПО, яким потрібна допомога, то вони виступають акторами, яких можна об'єднати в одну категорію користувач. Основними об'єктами є оголошення, оскільки саме над ними виконуються основні операції.

Системою можна користуватись, як не реєструючись, так і авторизувавшись. Це дещо змінюватиме функціонал, проте залишатиме можливість пошуку серед доступних оголошень.

Користувач - тут розглядається, як звичайний користувач системи, що був авторизований і якому доступний ширший функціонал.

Неавторизований користувач - будь-який користувач, що не зареєстрований або не увійшов в систему.

Під постами маються на увазі повідомлення, що містять тему, опис, місце (населений пункт чи область), категорію тощо.

Під поняттям “пошук по оголошеннях” можна розуміти пошук по вмісту слова у назві, описі, а також пошук шляхом фільтрації (вибір, наприклад, категорії).

На рис. 1.7 наведено схему варіантів використання, що детальніше описана в таблиці 1.1.

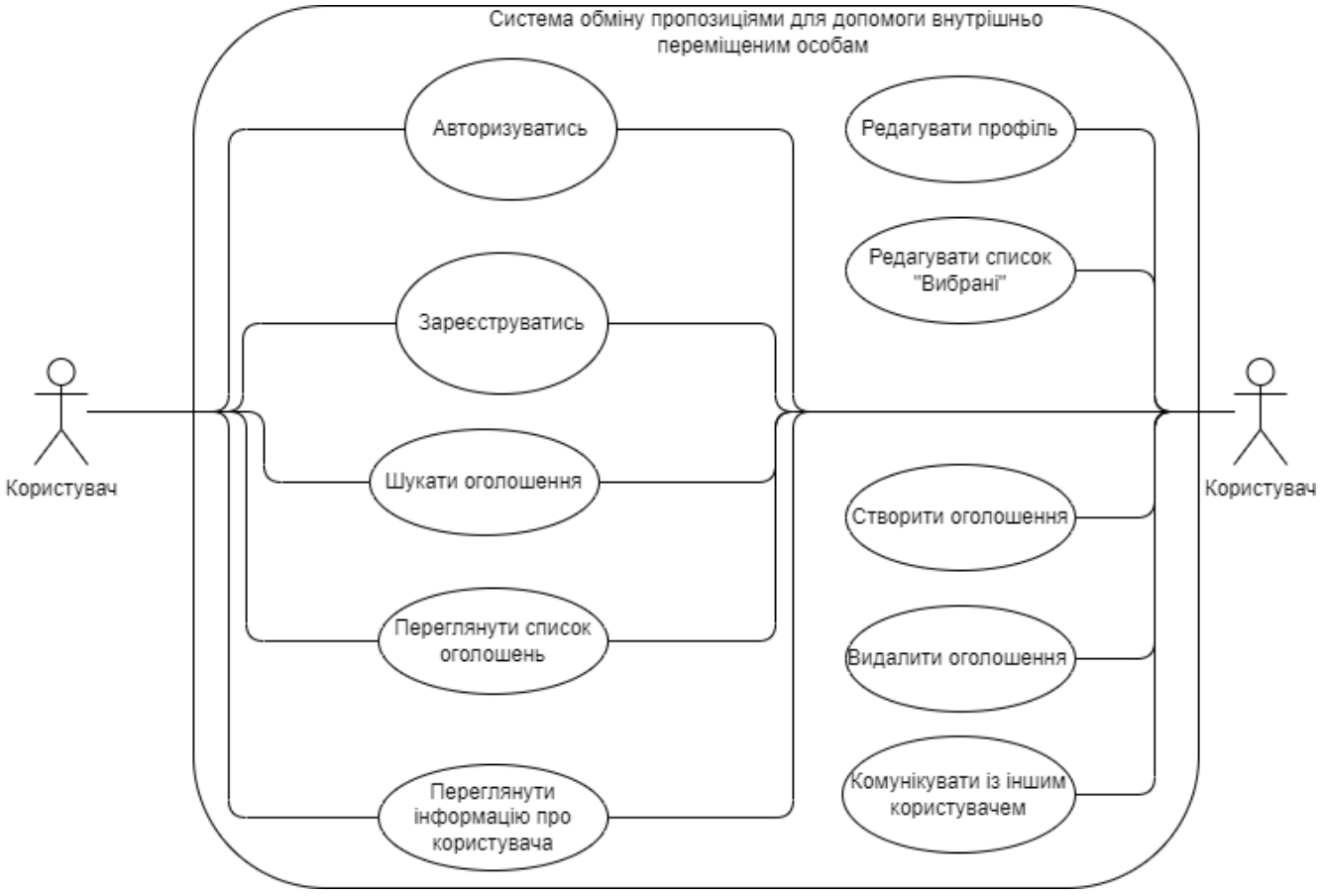


Рисунок 1.7 – Діаграма варіантів використання

Таблиця 1.1 – Функціональні вимоги

Актор	Назва функції	Функціональна вимога	Пріоритет
Неавторизований користувач, користувач	Перегляд списку оголошень	Система має надавати можливість перегляду списку та вмісту постів (оголошень), що було створено та розміщено користувачами. Оголошення відображаються в порядку “від найновіших”.	Головний
Неавторизований користувач	Авторизація	Авторизація користувача у системі	Головний
Неавторизований користувач	Реєстрація	Реєстрація користувача в системі	Головний
Неавторизований користувач, користувач	Пошук	Система має надавати користувачу можливість здійснювати пошук потрібних оголошень по назві, а також по категорії чи місцю, в якому це оголошення актуальне.	Головний
Користувач	Видалення оголошення	Система має дозволяти видаляти оголошення, що вже не актуальні.	Головний

Продовження таблиці 1.1

Актор	Назва функції	Функціональна вимога	Пріоритет
Користувач	Створення оголошення	Система має надавати можливість користувачу створювати нове оголошення (пропозицію допомогти чи пост про потребу в допомозі) при цьому вказуючи категорію, населений пункт, додавати опис.	Головний
Користувач	Редагування профілю	Система має дозволяти додавати певну інформацію про себе (наприклад, місто).	Другорядний
Користувач	Комунікація з іншим користувачем	Система має надавати можливість комунікувати (спілкуватись) з іншим користувачем шляхом текстових повідомлень.	Другорядний
Неавторизований користувач, користувач	Перегляд інформації про користувача	Система має надавати можливість переглянути дані, про користувача (власника оголошення) та список створених ним постів.	Другорядний

Кінець таблиці 1.1

Актор	Назва функції	Функціональна вимога	Пріоритет
Користувач	Редагування списку “Вибрані”	Система має надавати змогу додати оголошення, які користувач бажає “зберегти” у список “Вибрані”. У користувача повинна бути можливість переглянути цей список, та видалити оголошення з нього, якщо вони більше його не цікавлять.	Другорядний

1.4.2 Розроблення нефункціональних вимог

До нефункціональних вимог даного додатку можна віднести ті, що дозволять забезпечити користувачу легкість використання та приємний досвід користування. Було виділено наступні пункти:

– простий та інтуїтивний візуальний інтерфейс, швидкий доступ до “найпопулярніших дій”;

– захист від некоректних дій користувача шляхом обмеження, наприклад, діапазону вхідних значень та зрозуміле відображення помилок;

– наявність підказок для користувача, щодо призначення кнопок чи полів введення, допустимих форматів тощо;

– одна сторінка має відповідати одному функціоналу, щоб полегшити використання.

Крім того, варто виділити вимоги щодо вмісту оголошення. Для того, щоб користувач міг з легкістю розуміти вміст і посил повідомлення, в оголошення повинен бути заголовок. З такою ж ціллю варто виділити категорію (як це роблять в онлайн-магазинах), щоб можна було легко зорієнтуватись, в чому саме надається

допомога. Ще одним важливим елементом є місце, в якому пропозиція актуальна. Такий вигляд оголошення є необхідним для забезпечення приємного користувацького досвіду.

1.5 Постановка комплексу завдань модуля

1.5.1 Призначення розробки

Призначенням розробки є створення системи для допомоги ВПО (та людям, що теж можуть потребувати допомоги). Вона має дозволити налагодити зв’язок між користувачем, що має бажання чи змогу допомогти та людиною, що її потребує. Перший зможе розмістити оголошення про послуги чи речі, другий – про необхідність у них. Пошук по постах дозволить користувачам швидше надавати допомогу.

Окрім основної функції даної системи можна, наприклад, розмістити посилання на інші потрібні ресурси, що зробить систему більш корисною і частіше відвідуваною. Наприклад, додати в оголошення категорії житло посилання на ресурс “Прихисток”.

1.5.2 Цілі та задачі розробки

Основна ціль даної розробки полягає у спрощенні процесу пошуку шляхів можливої допомоги для людей, що цього потребують, використовуючи дошку оголошень, що дозволить зручно переглядати доступні варіанти. Водночас система, що розробляється має надати можливість зворотню – розміщати інформацію про потребу у зручній формі для пошуку.

Для того, щоб досягнути поставлених цілей потрібно розв'язати виділені задачі:

– визначити формат вхідних даних;

– створити функцію, що відповідатиме за можливість розміщення оголошення ;

– створити функцію, що забезпечуватиме пошук;

– створити функцію, що дозволить комунікувати з власником оголошення;

– створити функції, що відповідатимуть за перевірку правильності введеного;

– спроєктувати інтерфейс системи;

– розробити інтерфейс для забезпечення користувачу створених функцій;

– провести тестування.

1.6 Висновки до розділу

В результаті роботи над даним розділом було проведено дослідження предметної області. На цю мить, враховуючи вищеописану інформацію, можна сказати, що створення даного програмного забезпечення є актуальним.

Досліджено готові рішення, що мають аналогічне призначення. Визначено, що вони або частково покривають вимоги, або обмежують пошук, наприклад тільки категоріями чи містом.

Визначено основні функціональні вимоги, що сформувати основні можливості даної системи такі як створення та розміщення нового оголошення із темою, місцем, описом та категорією, пошук шляхом full-text search та фільтрацією, видалення постів, додавання їх у вибране. Сформовано також нефункціональні вимоги.

Описано призначення даної системи та головні задачі розробки.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

У даному пункті розглянемо основні процеси, в яких діє користувач у даній системі включаючи додавання оголошення, пошук серед наявних в системі, комунікацію з іншими користувачами.

Нижче описується процес створення оголошення:

– для створення нового оголошення користувач повинен бути авторизованим у системі (пройшовши даний процес шляхом використання email та паролю);

– користувач заповнює форму, в якій описує параметри оголошення, натиснувши перед цим відповідну кнопку, що відкриває форму. Користувач вводить дані у всі необхідні поля, додає фото за бажанням. Відбувається перевірка правильності введених полів;

– користувач натискає на кнопку підтвердження – на даному етапі формується об’єкт з відповідними значеннями полів та відправляється запит, в результаті чого відбувається внесення відповідних записів до бази даних. У разі успішного виконання оголошення з’являється на головному екрані.

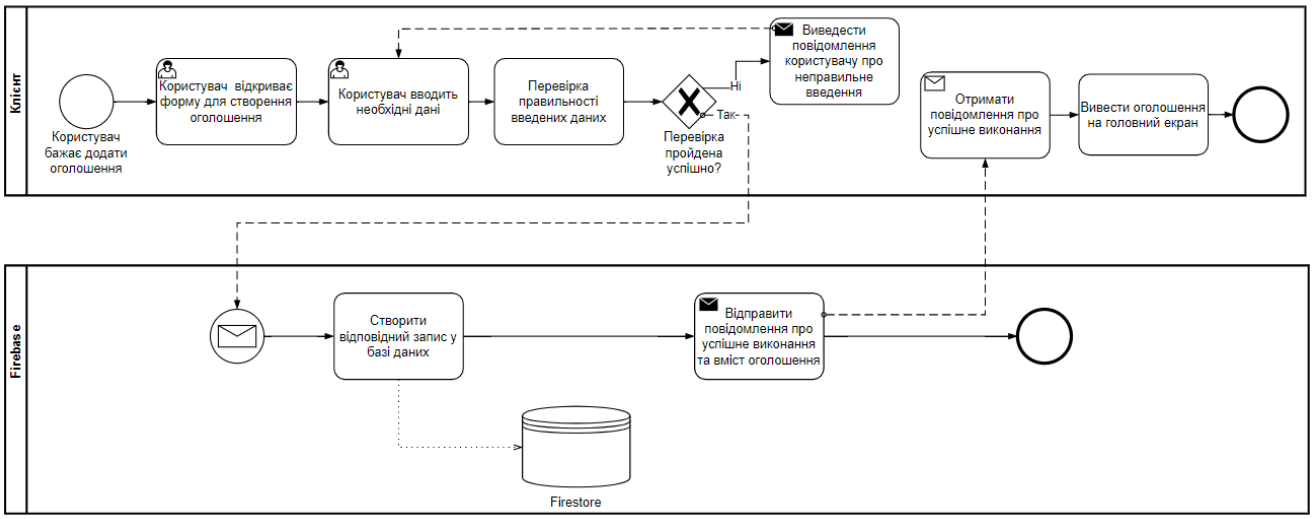


Рисунок 2.1 – BPMN-діаграма процесу додавання оголошення

Детальніше розглянемо процес пошуку:

– користувач в спеціально відведеному полі для пошуку вводить текст, який повинен міститись у полях оголошення;

– натискає кнопку пошуку і при цьому відбувається відправка запиту;

– відповідна функція опрацьовує запит здійснюючи full-text пошук по визначених полях (назва, категорія, опис);

– в результаті запиту повертаються список оголошень, що відповідають пошуковому запиту, у форматі JSON;

– записи відображаються на екрані у зручній для користувача формі.

Крім описаних вище можна розглянути інші, дещо менші процеси.

Процес авторизації відбувається шляхом введення електронної пошти та паролю.

Процес реєстрації відбувається шляхом введення чотирьох полів: електронна пошта користувача, ім’я, пароль та поле підтвердження пароля.

При виконанні обох процесів, якщо користувачем було введено невірні дані або дані, що не відповідають вимогам, користувач отримує відповідне повідомлення чи підказку.

Для того, щоб прокоментувати оголошення необхідно перейти на сторінку конкретного оголошення, та в полі для введення коментарів ввести повідомлення і натиснути відправити. Після цього коментар з’явиться на екрані.

2.2 Архітектура програмного забезпечення

Після аналізу предметної області та формування вимог визначено, що найкращим рішенням для досягнення поставлених задач буде реалізація системи у вигляді вебзастосунку. Такий вибір зумовлено тим, що це полегшить доступ до системи. Зважаючи на те, що ця система призначена для допомоги, то перешкоди мають бути мінімальними. При реалізації системи у вигляді вебзастосунку користувачу достатньо буде увійти на сайт, щоб отримати доступ. Крім того, доречність такої форми реалізації випливає і з проаналізованих готових програмних рішень у першому розділі даної роботи.

Оскільки розглядається створення системи, що актуальна і потрібна саме зараз, то було потрібне таке архітектурне рішення, що дозволило б спростити забезпечення інфраструктури сервера, в той самий час залишивши можливість реалізації необхідного функціоналу та забезпечення аналогічних показників якості.

В цьому випадку було обрано архітектурний шаблон Serverless.[3] В такому варіанті сервер насправді є, але управління серверною інфраструктурою надається обраним провайдером. Сервери автоматично розгортаються в кількох зонах для забезпечення високої доступності. Провайдер (постачальник послуг) у такому випадку надає сервіси, що допомагають формувати архітектуру застосунку. Саме ці сервіси відповідають за правильне опрацювання, безперебійний доступ до даних, та їх безпеку. Сам провайдер виступає сервісом для створення бекенду та хмарного зберігання даних (BaaS).

“Серверна” частина застосування безсерверної архітектури становить собою набір функцій, кожна з яких відповідає за лише одну дію. Застосунки безсерверної архітектури керуються подіями, тобто подія запускає функції, які здійснюють, наприклад операції з даними у базі даних. Створені розробником функції провайдер зберігає в хмарі.

В ролі провайдера було розглянуто Firebase та AWS Amplify, що є зараз одними з найпопулярніших при розробці застосунків з архітектурою Serverless. Нижче наведено таблицю, в якій описується порівняння цих двох варіантів.

Таблиця 2.1 – Порівняння Firebase та AWS Amplify

Характеристика	Firebase	AWS Amplify
Зберігання файлів	Cloud Storage	AWS S3
База даних	Realtime, Firestore	AWS AppSync
Аутентифікація користувача	+	+

Продовження таблиці 2.1

Характеристика	Firebase	AWS Amplify
Serverless функції	Cloud Functions	Lambda Function
Хостинг	+	+
Наявність плану Pay as you go	+	+
Безкоштовний ліміт	10 тис. аутент./місяць 2 млн запитів/місяць 400 тис. GB-seconds	1 млн запитів/місяць 400 тис. GB-seconds

Оскільки провайдери мають досить схожий функціонал і забезпечують необхідні функції, то вибір зроблено на основі безкоштовного ліміту, який не потребує оплати. Та й подальша оплата, завдяки можливості наданій провайдерами Serveless архітектури, буде стягуватись лише в залежності від використання.

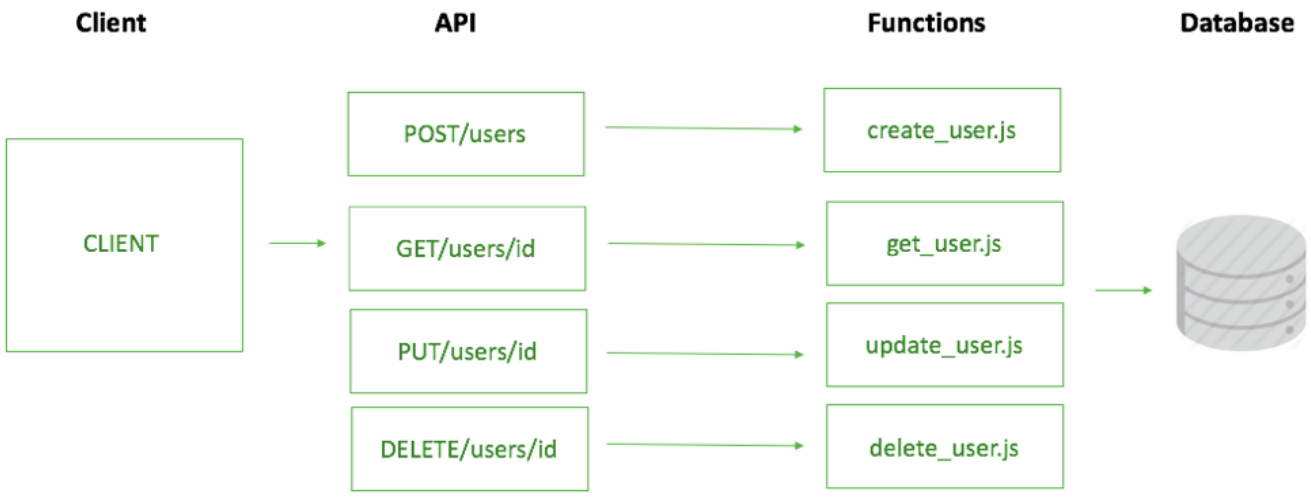


Рисунок 2.2 – Безсерверна архітектура

Отже, серверна частина буде побудовано за допомогою Firebase Functions, використовуючи Firestore в ролі бази даних. Доступ до бекенду буде надано завдяки сформованому API.[4] Таким чином кожна кінцева точка API буде відповідати конкретній функції та файлу.

Для написання функцій обрано Node.js – зручна платформа для роботи з JavaScript, яка виконує код поза браузером. Дозволяє створювати швидкі мережеві застосунки. Крім того, обрано фреймворк вебзастосувань Express, що забезпечує широкий спектр функцій для мобільних та веб застосунків.

Було прийнято рішення реалізовувати клієнтську частину застосунку з допомогою декларативної, ефективної та гнучкої JavaScript-бібліотеки React, що призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду — “компонент”[5].

Для розробки даного проєкту використовується редактор коду Visual Studio Code. Це зручний та добре налаштовуваний редактор, що розповсюджується безкоштовно. Дозволяє використовувати зручні розширення. Наприклад, було обрано до використання Prettier – зручний форматувальник коду, що аналізує код та при збереженні форматує його, враховуючи максимальну довжину рядка та за потреби обгортаючи код.

2.2.1 Firebase Firestore та Storage

Провайдер Firebase надає можливість зберігати дані потрібні для розроблюваного застосунку у двох видах баз даних. До них відносяться Realtime Database та Firestore. Обидва види цих баз даних забезпечують хмарне зберігання даних. Проте Firestore буде більш доцільною при складніших запитах, транзакціях чи сортуванні, в той час, як Realtime Database призначена скоріше для синхронізації даних з базовими запитами.

Обидві БД є NoSQL. Формат даних, який використовується для зберігання є різним. Realtime структурує дані в JSON дереві. Firestore надає можливість зберігати у формі, схожій до JSON, – документі, та організовувати ці документи у колекції. При цьому у кожного документа є id, яке можна задати або ж воно визначиться автоматично.

Оскільки синхронізація у даному конкретному випадку не є критичним аспектом, то прийнято рішення використовувати Firestore для організації зберігання даних.

Необхідно виділити те, які саме дані будуть зберігатись в базі. Головним об’єктом, дані про який необхідно зберігати, є оголошення. Зважаючи на аналіз предметної області, виділено такі необхідні для оголошення дані:

– назва або ж тема, що дозволить користувачам системи легко зрозуміти вміст оголошення;

– категорія, що полегшить пошук;

– опис, що дозволить детальніше описати пропозицію;

– місто, в якому пропозиція актуальна;

– відмітка, що відрізняє оголошення-пропозицію від оголошення, про потребу в допомозі;

– дата та час створення оголошення, що дозволить сортувати оголошення від найновіших;

– ім’я користувача, що розмістив оголошення.

Id кожного документа у цій колекції визначається автоматично самим Firebase.

Опис полів у колекції posts наведено у таблиці 2.2.

Таблиця 2.2 – Опис структури документів колекції posts

Поле	Тип	Призначення
title	string	Назва (заголовок) оголошення
category	string	Категорія
place	string	Місто
body	string	Опис
createdAt	string(ISO 8601)	Дата та час створення оголошення
userHandle	string	Ім’я користувача, що додав оголошення

Продовження таблиці 2.2

Поле	Тип	Призначення
likeCount	number	Кількість користувачів, що додали це оголошення у “Вибране”
isOffer	boolean	Прапорець, що позначає те, що оголошення є пропозицією

Окрім колекції з оголошеннями необхідно зберігати й інші дані. До списку колекцій, які будуть потрібними для роботи системи, входять:

– колекція для зберігання даних про зареєстрованих користувачів;

– колекція для зберігання даних про те, які оголошення користувачі додають у “Вибране”;

– колекція для зберігання коментарів до оголошення.

У таблиці 2.3 описано структуру документа у колекції users.

Таблиця 2.3 – Структура документів колекції users

Поле	Тип	Призначення
userId	string	Токен користувача
email	string	Електронна пошта користувача
handle	string	Ім’я користувача
createdAt	string(ISO 8601)	Дата, коли користувач зареєструвався
imageUrl	string	Посилання на фото користувача у Cloud Storage
location	string	Місто, в якому знаходиться користувач (не вимагається)
bio	string	Опис користувача (не вимагається)

Кінець таблиці 2.3

Поле	Тип	Призначення
phone	string	Номер телефону користувача

Таблиця 2.4 – Структура документів колекції favourites

Поле	Тип	Призначення
postId	string	Id оголошення, що було додане у “Вибране”
userId	string	Id користувача, що додав це оголошення у своє “Вибране”
postTitle	string	Заголовок оголошення
createdAt	string(ISO 8601)	Дата та час, коли оголошення було додано до колекції

Таблиця 2.5 – Структура документів колекції comments

Поле	Тип	Призначення
body	string	Вміст коментаря
createdAt	string(ISO 8601)	Дата та час створення коментаря
userId	string	Id користувача, що додав коментар
userHandle	string	Ім’я користувача, що додав коментар
userImage	string	Фото користувача
postId	string	Id оголошення, до якого додано коментар

У кожній із цих таблиць id документів визначається автоматично і мають такий формат: “4L9GhCkg0LAp7huLM4zY”. Для реляційної бази даних внесення

поля userHandle в документ колекції comments було б надлишковим. Таке рішення зумовлене особливістю зберігання даних та виконанням запитів у Firestore. Firebase не дозволяє створювати таких складних запитів об’єднуючи таблиці.

Для зберігання користувацьких медіа, таких як фото користувача обрано один із сервісів, що надається провайдером Firebase – Cloud Storage. У кожного файлу є шлях, за яким можна отримати його. Тому в колекціях достатньо зберігати лише посилання на цей медіавміст у Storage.

Цілісність даних збережених у Firestore забезпечує провайдер. Крім того, доступ до даних потрібно визначати за допомогою спеціальних правил. Вони є основним способом для розробника захистити доступ до БД, тому їх задання є важливим кроком при розробці з використанням баз даних Firebase. Приклад задання таких правил наведено на рис. 2.3.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
          //request.time < timestamp.date(2022, 6, 4);
    }
  }
}
```

Рисунок 2.3 – Приклад задання правил доступу для Firestore

Ще одним способом зробити доступ до даних безпечним є аутентифікація користувачів, що зможуть вносити зміни до колекцій.

2.2.2 Firebase Function

Для створення функцій, об'єднання яких і становитиме серверну частину застосунку використано один із сервісів, який надає провайдер Firebase. Firebase Functions виступають у ролі FaaS[6], та надають можливість керування вмістом бази даних і забезпечують виконання необхідної логіки. Дані функції запускаються кожен раз, коли відбувається якась подія. На цей момент доступні різноманітні

тригери, які можна прослуховувати та реагувати на них запуском функції. Серед доступних тригерів:

– тригери Realtime Database;
– тригери аутентифікації;
– тригери, що реагують на зміни в хмарному сховищі;
– HTTP тригери та інші.

Для того, щоб використовувати Firebase Functions потрібен спеціальний інтерфейс командного рядка Firebase CLI, установлений з використанням npm, спеціального менеджера пакунків для JavaScript. Далі з допомогою певних команд відбувається ініціалізація проєкту, після виконання яких у каталозі проєкту формується нова спеціально визначена структура. Її вигляд наведено на рис. 2.4.

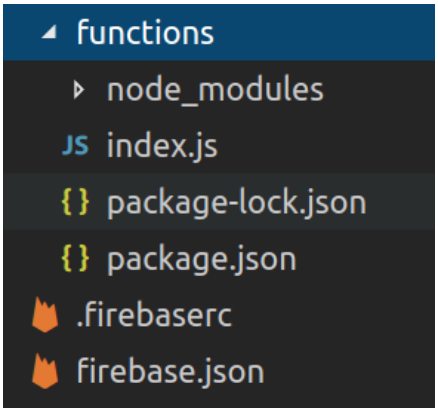


Рисунок 2.4 – Структура проєкту Firebase Functions

Файл firebase.json містить основну інформацію про створюваний проєкт, його властивості.

У новоствореній папці functions буде розміщено код створюваних функцій. В цій же папці розміщено файл package.json. Це файл пакета npm, що описує код. Папка node_modules містить усі встановлені залежності npm. Основним файлом "постачальником" програмного коду функцій виступає файл index.js.

Для того, щоб полегшити розуміння коду та відділити функції за призначенням було створено таку структуру папки functions:

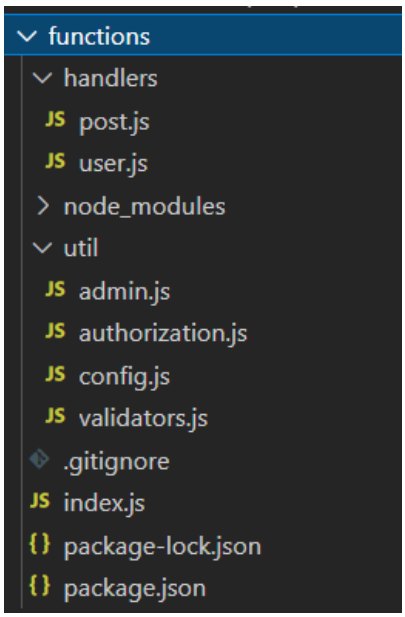


Рисунок 2.5 – Структура директорії functions

Таблиця 2.6 – Опис елементів серверної частини

Назва елементу	Призначення
handlers	Папка, що містить файли, з функціями, що дозволяють опрацьовувати запити пов’язані з певними колекціями, наявними у Firestore і реалізовують певну функціональну логіку.
util	Папка, що містить файли певного другорядного призначення, або ті, вміст яких використовується в інших як модулів.
admin.js	Файл, який містить необхідні модулі, такі як firebase-admin та експортує об’єкт db, забезпечує доступ до firestore.
authorisation.js	Файл, в якому міститься функція перевірки чи є користувач авторизованим. Результат виконання цієї функції використовується для перевірки доступності користувачу інших функцій (Наприклад, розміщення оголошень).

Продовження таблиці 2.6

Назва елементу	Призначення
config.js	Файл, що містить необхідні параметри для конфігурації, надана Firebase.
validators.js	Файл містить у собі три функції, що призначені для перевірки (валідації) даних, що поступили від клієнта. (Функції validateSignupData, validateLoginData, reduceUserDetails)
post.js	Даний файл містить у собі функції, що пов’язані із діями з оголошеннями. Вони забезпечують додавання, видалення та опрацювання даних у колекціях posts. favourites, comments. (Функції postOne, getOnePost, getAllPosts, likePost, unlikePost, deletePost, commentOnPost, search)
user.js	Даний файл містить у собі функції, що пов’язані із діями з користувачами. Вони забезпечують додавання, видалення та опрацювання даних у колекції users. (Функції signup, login, uploadImage, addUserDetails, getCurrentUser, getUserDetails)
index.js	Містить функції, що викликаються тригерами хмарного сховища. Об’єднує всі функції, що були створені та організовує їх для можливості виклику їх зі сторони клієнта. (Функції onPostDelete, onImageChange)

Кожна зі створених функцій досить коротка і відповідає лише за одну певну конкретно визначену задачу. Таким чином покриваючи весь бажаний функціонал. Детальну інформацію про кожну з функцій наведено у таблицях 2.7 та 2.8.

Таблиця 2.7 – Опис створених функцій (стосуються оголошень)

Назва функції	Опис
postOne	Функція, що відповідає за додавання нового оголошення. Створює новий запис у колекції posts з отриманими від клієнта темою, категорією, містом, описом, кількістю додавань до “Вибраного” рівною нулю, поточним часом та іменем користувача, що створив оголошення. Якщо виконання коректне, то повертає вміст оголошення у JSON форматі, якщо ні – помилку із повідомленням “Щось пішло не так!”
getOnePost	Функція, що відповідає за отримання одного конкретного оголошення із колекції post. Шукає документ із вмістом оголошення по Id, що було отримане від клієнта. В разі успіху повертає оголошення та всю пов’язану із ним інформацію. В іншому випадку повертає повідомлення “Такого оголошення не знайдено!”
getAllPost	Функція, що відповідає за отримання документів із колекції posts. Вміст документів (оголошень) повертає у JSON форматі.
getAllNeeds	Функція, що дозволяє отримати всі оголошення (в яких просять про допомогу).
getAllOffers	Функція, що дозволяє отримати оголошення (пропозиції).
search	Функція, що відповідає за реалізацію full-text пошуку. На вхід приймає текстове поле, на відповідність якому буде здійснюватись перевірка. повертає клієнту вміст оголошень, що відповідають пошуковому запиту.

Продовження таблиці 2.7

Назва функції	Опис
likePost	Функція, що відповідає за додавання оголошень авторизованим користувачем у колекцію “Вибране”. У функцію надходить ідентифікатор оголошення, а в результаті авторизації отримується ідентифікатор користувача. Перевіряється вміст такого оголошення у колекції posts. Якщо оголошення не знайдено повертається повідомлення “Оголошення не знайдено!” Відбувається перевірка на вміст запису з такими полями у колекції favourites. Якщо такий запис вже є, то додавання не відбувається, в іншому випадку відповідний документ додається до колекції, збільшується лічильник додавань до “Вибраного” в оголошення, повертається вміст оновленого оголошення у форматі JSON.
unlikePost	Функція, що відповідає за видалення оголошень користувачем із колекції “Вибране”. Виконання допустиме лише для авторизованих користувачів. Логіка функції аналогічна попередній. Відмінність полягає у зменшенні лічильника.
deletePost	Функція, що відповідає за видалення оголошення. Доступна лише для авторизованого користувача, власника оголошення. Можливими результатами виконання даної функції можуть бути різного роду повідомлення: “Оголошення не знайдено!”, “Ви не авторизовані!”, “Оголошення видалено успішно”.
commentOn Post	Функція, що відповідає за можливість коментувати оголошення. Доступна авторизованим користувачам. Приймає від клієнта ідентифікатор оголошення та вміст коментаря. Перевіряє чи вміст коментаря не пустий. Додає відповідний документ до колекції comments. Повертає створений документ коментаря у форматі JSON.

Таблиця 2.8 – Опис створених функцій (стосуються користувача)

Назва функції	Опис
signup	Функція, що відповідає за реєстрацію нового користувача. Викликає функцію перевірки введених даних. Використовує функцію для створення користувачів, що надана Firebase Authentication, що повертає спеціальний токен доступу користувача. Додає запис у колекцію users. Повертає токен доступу або повідомлення про помилки.
login	Функція, що відповідає за вхід користувача в систему. Викликає функцію перевірки введених даних. Використовує функцію для створення користувачів, що надана Firebase Authentication, що повертає спеціальний токен доступу користувача. Повертає отриманий токен доступу або повідомлення про помилки.
uploadImage	Дозволяє змінити фото користувача зі стандартного, на обраний файл. Формати зображення, які може приймати функція: jpeg та png. Обраний користувачем файл зберігається у хмарному середовищі Cloud Storage. Для роботи з файлами використовується спеціальний модуль node.js busboy. В разі успішного завантаження, посилання на зображення в колекції користувачів змінюється.
addUserDetails	Функція дозволяє оновити профіль користувача, додавши місто, в якому активний користувач чи опис про себе. Може приймати як обидва поля, так і лише одне із них. Для того, щоб правильно заповнити необхідні поля використовується функція reduceUserDetails. В разі успішного виконання редагує колекцію users та повертає повідомлення “Профіль оновлено успішно”.
getUser Favourites	Функція, що повертає назви оголошень, які користувач додав до “Вибраного”.

Кінець таблиці 2.8

Назва функції	Опис
getCurrent User	Функція дозволяє отримати всю доступну інформацію про поточного користувача. Дані повертаються у форматі JSON, включно з колекцією “Вибране” даного користувача.
getUser Details	Функція дозволяє отримати інформацію про вибраного користувача за його ідентифікатором. Дані повертаються у форматі JSON, включно з колекцією оголошень даного користувача.
Validate SignupData	Функція перевірки правильності введених користувачем даних для реєстрації. Перевіряється чи не були поля пустими, чи відповідає пошта шаблону.
validateLogin Data	Функція перевірки правильності введених користувачем даних для реєстрації. Перевіряється чи не були поля пустими.
onUserImage Change	Функція, що викликається тригерами хмарного сховища. Реагує на оновлення інформації в колекції users. Відбувається перевірка того, чи було змінено посилання на інше фото. Якщо посилання змінено, то необхідно змінити посилання на зображення користувача у коментарях.
onPostDelete	Функція, що викликається тригерами хмарного сховища. Реагує на видалення інформації в колекції posts. Якщо оголошення видалено, то видаляються й відповідні записи у колекції favourites.

2.2.3 Firebase Authentication

Оскільки деякі з представлених функцій можуть бути виконаними лише для авторизованих користувачів, то потрібно певним чином управляти даними

користувачів. Для таких цілей обраний провайдер Firebase пропонує один зі своїх сервісів.

Firebase Authentication надає набір функцій для аутентифікації користувачів, що дозволяють, передавши лише дані введені користувачем, провести реєстрацію користувача чи його перевірку при вході в систему.

Крім того, Firebase Authentication дозволяє програмі безпечно хмарно зберігати дані користувача та інтегрується з іншими службами Firebase. Можна проводити аутентифікацію користувачів за допомогою адреси електронної пошти та паролю. Firebase Authentication SDK надає методи створення та керування користувачами, які використовують свої адреси електронної пошти та паролі для входу.

Іншим способом аутентифікації є номер телефону: для цього користувачу надсилають СМС-повідомлення. Можна також інтегрувати постачальників аутентифікації таких, як Google, Apple, Facebook, Twitter, GitHub.

В даному проєкті було обрано метод аутентифікації через електронну пошту та пароль. Для цього було використано набір функцій із категорії Password Authentication у документації Firebase.[7]

Для того, щоб створити новий обліковий запис потрібно спочатку отримані дані від форми реєстрації застосунку перевірити, наприклад на відповідність складності пароля, на правильність введення електронної пошти. Далі адреса електронної пошти та пароль передаються у спеціальну функцію createUserWithEmailAndPassword, яка в разі успішної реєстрації повертає спеціальний токен, що слугує для авторизації. Якщо новий обліковий запис створено, користувач автоматично входить в систему.

Для того, щоб виконати вхід користувача в систему необхідно передати адресу електронної пошти та пароль користувача у функцію signInWithEmailAndPassword. В разі успішного виконання буде повернено спеціальний токен доступу.

Таким чином після першого входу користувача створюється обліковий запис пов’язаний з даними за допомогою яких користувач увійшов. Цей новий обліковий

запис зберігається як частина проєкту Firebase.

Також при розробці системи було використано вбудований метод Firebase для перевірки маркерів входу (токенів). Отриманий токен від клієнта декодується функцією verifyIdToken та в разі, якщо токен правильний та актуальний, то функція повертає декодований токен, який надалі можна використовувати для авторизації.

Задля безпеки зберігання даних користувача Firebase Authentication “використовує внутрішньо модифіковану версію scrypt для хешування паролів”.[8] Scrypt – це криптографічна функція для формування ключа, використовуючи пароль, що потребує більше часу для обчислення та вимагає інтенсивних обчислювальних процесів.[9] Це робить такий спосіб зберігання облікових даних добре захищеним.

2.2.4 Algolia

В ході реалізації системи необхідно реалізувати функцію пошуку по текстовому вмісту документів колекцій. Але зважаючи на особливості реалізації запитів до бази даних, організувати full-text пошук неможливо. Єдиним варіантом пошуку, який може бути реалізованим без інтеграції сторонніх сервісів, є спосіб, що дозволить здійснювати пошук лише по першому слову в одному з полів.

Для такого завдання на офіційному сайті Firebase, пропонують інтегрувати один зі сторонніх пошукових сервісів:

– Elastic;
– Algolia;
– Typesense.

В даній роботі було обрано використовувати пошуковий сервіс Algolia. Він простий у застосуванні, наявний у Firebase у вигляді розширення, що спрощує його додавання до проєкту. Крім того, Algolia пропонує вже готові компоненти для створення інтерфейсу користувача, наприклад, з можливістю instant search.

Розширення Algolia додає автоматично створені тригери на обрані колекції даних та в разі оновлення даних, змінює відповідні індекси у своєму сховищі. Коли здійснюється запит на пошук, то відбувається звернення до індексів і пошук по них.

За допомогою вебсайту даного сервісу можна визначити поля, що додаватимуться до індексу, поля по яких можна здійснювати пошук, сортування чи фільтрацію, що є дуже зручним при реалізації відповідних функцій застосунку, що створюється в ході виконання роботи.

```
const algoliasearch = require("algoliasearch");
const algolia = algoliasearch();

exports.search= (req,res)=>{
  const query = req.query.s;
  const client = algoliasearch('9QY4355JTC',
                                [redacted]);
const index = client.initIndex('posts');

  index
  .search(query)
  .then(({hits}) => {
    console.log(hits);
    return res.json(hits);
  }).catch(err => console.error(err));
}
```

Рисунок 2.6 – Вигляд Firebase функції search з використанням Algolia

Algolia забезпечує безпеку створених індексів. “Сервери Algolia підтримують шифрування HTTPS, обмін криптографічними ключами Діффі-Хеллмана з еліптичною кривою, підписаними RSA та ECDSA, а для підтримуваних клієнтів також ідеальні методи прямої секретності (PFS), щоб захистити трафік від скомпрометованого ключа або криптографічного прориву. Algolia використовує лише стандартизовані технології шифрування.”[10]

2.2.5 REST API

Як було вказано у попередніх пунктах, Serverless застосунки є керовані подіями. Ці події роблять виклик функцій, що були написані. Такими подіями для Firebase Function можуть бути, наприклад, тригери хмарних сховищ (було використано при реалізації). Іншим видом тригерів, що запускають функції на виконання є HTTP-тригери. Хоч для HTTP запитів до firebase функцій API шлюз не

обов’язковий, але це дозволяє обгорнути всі функції в одну. Саме такий варіант реалізації було обрано для розробки.

У даній роботі обрано архітектурний підхід REST для організації API. Для того, щоб забезпечити доступ до деяких запитів лише авторизованим користувачам використовується метод при якому до хедеру запиту додається параметр Authorization: “Bearer <token>”. Нижче у таблиці 2.9 наведено список доступних запитів.

Таблиця 2.9 - Документація створеного API

Метод	Шлях	Дія
GET	/posts	Доступний для неавторизованих користувачів, повертає оголошення з бази даних. (getAllPosts)
GET	/post/{postId}	Доступний для неавторизованих користувачів. Повертає вміст конкретного оголошення. (getOnePost)
GET	/search	Доступний для неавторизованих користувачів. За значенням текстового параметра здійснює пошук. Повертає вміст оголошень, що містять в назві чи в описі даний параметр.(search)
POST	/post	Доступно лише авторизованим користувачам. Для запиту потрібен токен. Дозволяє додати оголошення в базу даних. (postOne)
POST	/post/{postId}/like	Доступно лише авторизованим користувачам, дозволяє додати оголошення у “Вибране”. (likePost)

Продовження таблиці 2.9

Метод	Шлях	Дія
POST	/post/{postId} /unlike	Доступно лише авторизованим користувачам, дозволяє видалити оголошення зі списку “Вибране”. (unlikePost)
POST	/post/{postId} /comment	Доступно лише авторизованим користувачам, дозволяє додати коментар до оголошення. (commentOnPost)
DELETE	/post/{postId}	Доступно лише авторизованим користувачам, що є власниками оголошень. Дозволяє видалити оголошення. (deletePost)
GET	/posts/need	Повертає список оголошень про потребу в допомозі (getAllOffers)
GET	/posts/offer	Повертає список оголошень-пропозицій (getAllNeeds)
POST	/signup	Дозволяє зареєструватись новому користувачу, повертає токен. (signup)
POST	/login	Використовується для входу користувача в систему, повертає токен.(login)
POST	/user/img	Доступно лише авторизованим користувачам, дозволяє завантажити нове зображення користувача. Повертає повідомлення про успішне виконання. (uploadImage)
GET	/user/fav	Дозволяє отримати список “Вибране” поточного користувача (getUserFavourites)
GET	/user	Доступно лише авторизованим користувачам, використовується для отримання інформації про поточного користувача. (getCurrentUser)

Кінець таблиці 2.9

Метод	Шлях	Дія
POST	/user	Доступно лише авторизованим користувачам, використовується для редагування профілю користувача (додавання нових даних). Повертає повідомлення про успішне виконання. (addUserDetails)
GET	/user/{userId}	Дозволяє отримати інформацію про користувача, список його оголошень.(getUserDetails)

2.3 Висновки до розділу

В ході написання даного розділу було виділено основні процеси, описано їх послідовність виконання та проведено моделювання процесів.

Для реалізації визначених завдань було прийнято рішення реалізовувати систему у вигляді вебзастосунку. Таке рішення зумовлено зручністю та легкою доступністю для користувачів, що виявлять бажання допомогти або ж потребують допомоги. Вебзастосунок не потрібно завантажувати, достатньо мати доступ до мережі Інтернет.

Описано архітектурний підхід, який було обрано для побудови системи. Serverless архітектура дозволяє витрачати менше зусиль для підтримки сервера. Ці зобов’язання бере на себе провайдер.

У ролі провайдера (BaaS) було розглянуто два варіанти Firebase та AWS Amplify. Кінцевим вибором став перший, в основному через надання всіх необхідних функцій та більший обсяг безкоштовних ресурсів.

Клієнтську частину застосунку прийнято рішення реалізовувати з допомогою бібліотеки React.

В ролі бази даних обрано Firestore для збереження необхідних даних про користувачів та оголошення та Storage для зберігання користувацьких медіа.

Оскільки Firestore є NoSQL базою даних, то було описано структуру колекцій та документів.

Наведено опис створених Firebase функцій, завдяки яким відбувається робота з колекціями та опрацьовуються запити з клієнтської частини. Створені функції дозволяють покрити функціональні вимоги, що описані у розділі 1. Для вирішення завдання аутентифікації користувачів використано сервіс наданий провайдером - Firebase Authentication. Для вирішення завдання повнотекстового пошуку до проєкту було інтегровано пошуковий сервіс Algolia.

Всі створені функції було обгорнено в REST API. Список можливих запитів наведено у таблиці 2.9.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

Перед процесом реалізації клієнтської частини застосунку було проведено проєктування зовнішнього виду графічного інтерфейсу. Розроблюваний дизайн має забезпечувати користувачу хорошу використовність (usability).

У своїй статті Якоб Нільсен, провідний консультант по web-usability, доктор наук наукового напрямку людино-комп'ютерної взаємодії Технічного Університету Данії в Копенгагені, зазначає, що використовність можна розглядати як атрибут якості, який оцінює, наскільки простими є користувацькі інтерфейси у використанні. Одними з показників якості використовності є те, наскільки легко користувачу, що вперше стикнувся з інтерфейсом, зрозуміти як ним користуватись та виконувати необхідні дії, та ефективність – те, як швидко користувач може виконати необхідну дію.[11]

Крім того, варто враховувати для якої групи користувачів і для яких цілей розробляється даний вебзастосунок. Основна ціль даної розробки полягає у наданні людям ресурсу, що дозволить запропонувати чи знайти допомогу. Тому основні функції мають бути легко доступними, а час пошуку необхідних кнопок, вкладок та інших функціональних компонентів мінімальним. При першому використанні у користувача не повинна зникати “мотивація” до користування застосунком.

Враховуючи вимоги що описані у розділі 1 та вищесказане, було прийнято рішення спершу визначитись, які саме сторінки потрібні, як організувати зв’язок між ними. Визначено такі сторінки:

– головний екран, на якому відображатимуться оголошення та вся інформація про них. На цьому екрані повинне бути розміщеним поле для пошуку;

– екран входу;

– екран реєстрації;

– екран для відображення інформації про користувача, включно з відображенням його оголошень;

– екран для відображення оголошень, які користувач додав до “Вибраного”;

– екран для відображення повної інформації про оголошення, включно з коментарями.

Для того, щоб спроєктувати вигляд інтерфейсу користувача було створено прототип вебзастосунку. Прототип – це модель застосунку, що візуалізує макет елементів та функцій. Така модель є дуже корисною для розуміння того, як в кінцевому результаті буде виглядати інтерфейс користувача, та дозволяє внести зміни ще на етапі проєктування.

Для створення прототипу було використано потужний графічний інструмент для розробки та прототипування інтерфейсів Figma. Перевагами цього інструменту є можливість використовувати безкоштовно, наявність вебверсії, зберігання файлів користувача у хмарі.

Було обрано основні кольори застосунку, шрифти, створено прототипи основних екранів.

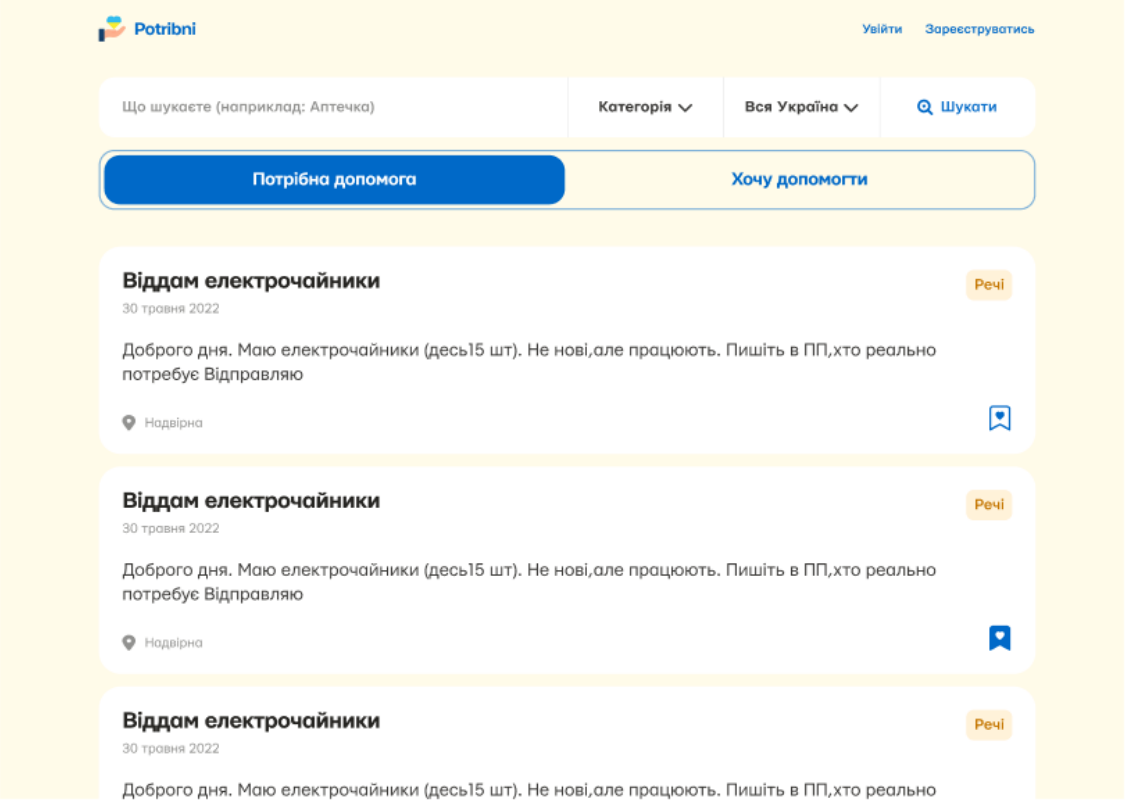


Рисунок 3.1 – Головний (початковий) екран

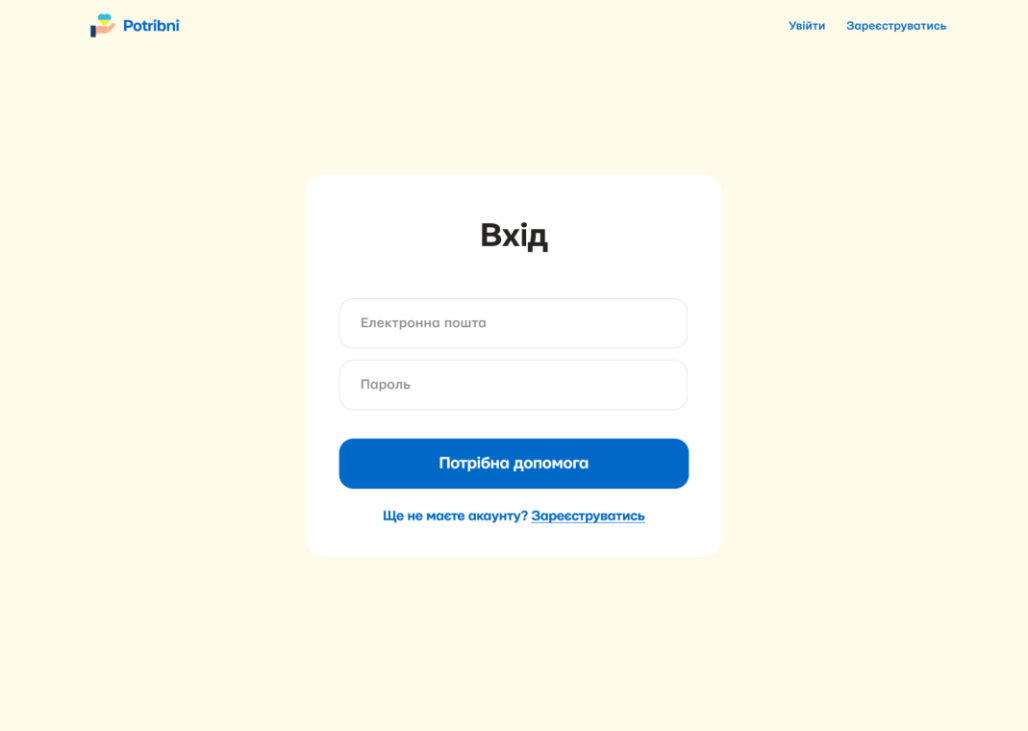


Рисунок 3.2 – Екран входу

3.1 React

В процесі роботи над системою було прийнято рішення реалізовувати вебзастосунок у вигляді Single Page Application[12]. Такий тип вебзастосунків є швидшим, адже не становить собою набір сторінок, що змінюють одна одну при надсиланні запитів на сервер. Односторінкові вебзастосунки фактично являють собою одну сторінку, компоненти якої динамічно змінюються залежно від дій користувача. В той час як багатосторінкові – завантажують цілі сторінки.

Всі елементи сторінки завантажуються одразу після ініціалізації. Коли завантажується нова інформація в SPA компоненти не змінюються повністю, а лише частково. Такий спосіб організації дозволяє пришвидшити процес надання відповіді на дії користувача та зменшити об’єм даних, які потрібно передавати.

Прикладами односторінкових застосунків є чимало відомих програмних рішень, якими користувачі користуються ледь не щодня: Facebook, Twitter, Gmail, Google Maps та інші.

На цей момент існує різноманіття JavaScript фреймворків та бібліотек для створення Single Page Applications. Одними з найпопулярніших є AngularJS, VueJS, ReactJS, EmberJS, BackboneJS.

Для даного проєкту було обрано JavaScript бібліотеку React. “React спрощує створення інтерактивних інтерфейсів. Потрібно лише описати, як різні частини інтерфейсу виглядають у кожному стані застосунку і React ефективно оновить та відрендерить лише потрібні компоненти, коли дані зміняться.”[13]

Для того, щоб створити односторінковий вебзастосунок з допомогою React було використано Create React App. Це офіційно підтримуваний спосіб створення SPA з допомогою обраної бібліотеки.

Згідно з офіційною документацією React create-react-app “встановлює осередок для розробки таким чином, щоб використовувати найновіші можливості JavaScript, робить розробку комфортнішою, а також оптимізує додаток для продакшну.”[13]

Рисунок 3.3 – Структура проєкту одразу після створення з допомогою create-react-app

Новостворений проєкт містить дефолтний набір папок та файлів серед яких є і як файли конфігурації, так і, наприклад, файл з іконкою. Основними автоствореними файлами є package.json, що містить інформацію про застосунок та index.js, що є вхідною точкою JavaScript.

В ході виконання проєкту структуру папок та файлів було змінено наступним чином:

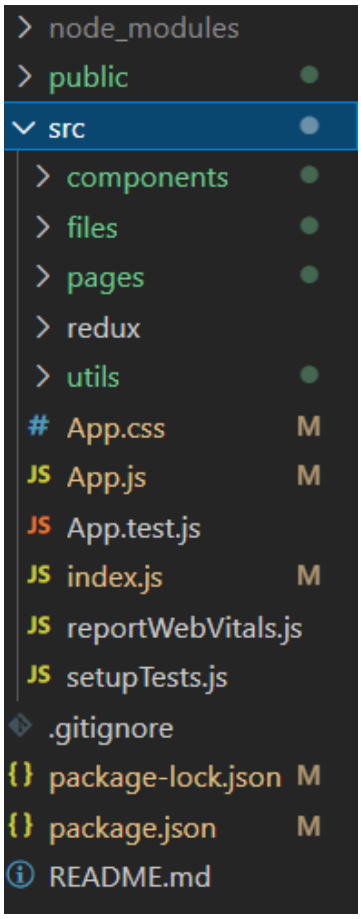


Рисунок 3.4 – Структура клієнтської частини застосунку

Таблиця 3.1 – Структура теки src

Назва елементу	Опис елементу
App.js	Файл, що описує основний компонент DOM дерева, що сприймається як вхідна точка.
App.css	Основний файл з описом стилів для елементів застосунку.

Продовження таблиці 3.1

Назва елементу	Опис елементу
index.js	Вхідна точка JavaScript.
components	Папка, в якій зберігаються всі створені компоненти, що складатимуть вигляд екранів. Наприклад, кнопки, форми, діалоги тощо.
files	Папка, що містить зображення чи іконки використані в застосунку.
pages	Папка, в якій містяться компоненти вищого рівня і використовуються в ролі екранів (Основний екран, екран входу, реєстрації тощо).
utils	Папка, що містить файли з іншими допоміжними часто використовуваними даними.
redux	Папка, в якій зберігаються файли пов’язані з роботою бібліотеки Redux.

У наступній таблиці наведено список та опис створених компонентів для реалізації вимог до системи та відповідності створеному прототипу.

Таблиця 3.2 – Опис компонентів клієнтської частини

Назва компонента	Опис компонента
Home	Компонент, що відповідає за відображення головного екрану. Складається з багатьох інших компонентів, наприклад, для відображення оголошень, спеціальних кнопок тощо.
Login	Компонент, що відповідає за відображення форми для входу в систему. Містить поля для введення пошти та паролю і кнопку підтвердження.

Продовження таблиці 3.2

Назва компонента	Опис компонента
Signup	Компонент, що відповідає за відображення екрана реєстрації. Містить поля для введення пошти, паролю, імені та кнопку підтвердження.
User	Компонент, що відповідає за відображення екрана користувача включно з інформацією про нього та оголошення, розміщені ним.
CommentForm	Компонент, що відповідає за відображення форми для внесення коментарів.
Comments	Компонент, що відповідає за відображення наявних коментарів.
DeleteButton	Компонент-кнопка, що відображається власникам оголошень і дозволяє видалити опубліковане оголошення.
BookmarkButton	Компонент-кнопка, що відповідає за можливість додавання оголення в колекцію “Вибране”. Змінює вигляд залежно від того, додано чи ні оголошення в колекцію.
AddPost	Компонент-кнопка, що відповідає за відображення форми для внесення нового оголошення. Містить необхідні елементи для введення даних про нього.
Edit	Компонент-форма для внесення змін до профілю користувача (додавання опису, міста…)
ProfileSkeleton	Компонент, що відповідає за відображення скелета елемента під час його завантаження. Скелет блоку з інформацією про користувача.
PostSkeleton	Компонент, що відповідає за відображення скелета елемента під час його завантаження. Скелет компонента для відображення оголошення.

Кінець таблиці 3.2

Назва компонента	Опис компонента
Post	Компонент, що відповідає за відображення кожного з оголошень на екрані. Відображає назву, категорію, локацію, опис та час створення оголошення.
PostDialog	Відповідає за відображення діалогу, в якому відображається детальніша інформація про оголошення.
Profile	Компонент, який дозволяє відобразити інформацію про користувача. Не є окремим екраном, а лише складовою частиною екрана користувача.
Navbar	Компонент, що відповідає за відображення верхньої панелі навігації, що змінюється залежно від того, авторизований користувач чи ні. Дозволяє перейти на основний екран, екран реєстрації, входу тощо.
SearchBar	Відповідає за відображення пошукової панелі, включно зі спадними меню для вибору міста та категорії.

3.2 Redux

Для управління станом програми JavaScript разом з React використовують бібліотеку Redux. Redux допомагає оптимізувати код та забезпечує редагування коду під час налагодження.

Оскільки вимоги до односторінкових застосунків зростають в міру їх вдосконалення, то потрібно брати на себе управління великою кількістю станів, використовуючи JavaScript. Зі збільшенням об’єму застосунку таке управління стає все складнішим.

На противагу вищесказаному, Redux допомагає упорядкувати зміни станів та зробити їх передбачуваними, за допомогою певних правил, що обмежують те, коли і яким чином зміни можуть відбутись.

Основний зміст цих правил відображено у трьох принципах, що вказані у документації Redux[14]:

– “глобальний стан програми зберігається в дереві об'єктів в межах одного сховища”;

– “єдиний спосіб змінити стан - виконати дію. Дії (actions) – це прості об’єкти, що описують те, що сталося в програмі, і служать єдиним способом описати намір змінити дані”;

– “щоб указати, як дерево станів перетворюється за допомогою дій, використовуються чисті редьюсери (reducers)”.

Оскільки в Redux можуть лише опрацьовуватись прості дії-об’єкти, reducer повинен бути чистою функцією, то для обробки дій-функцій і запитів до бази даних необідно використовувати проміжне програмне забезпечення. В даному проєкті в ролі цього ПЗ було використано redux-thunk.

Таким чином архітектура застосунку із застосуванням бібліотеки React виглядатиме наступним чином:

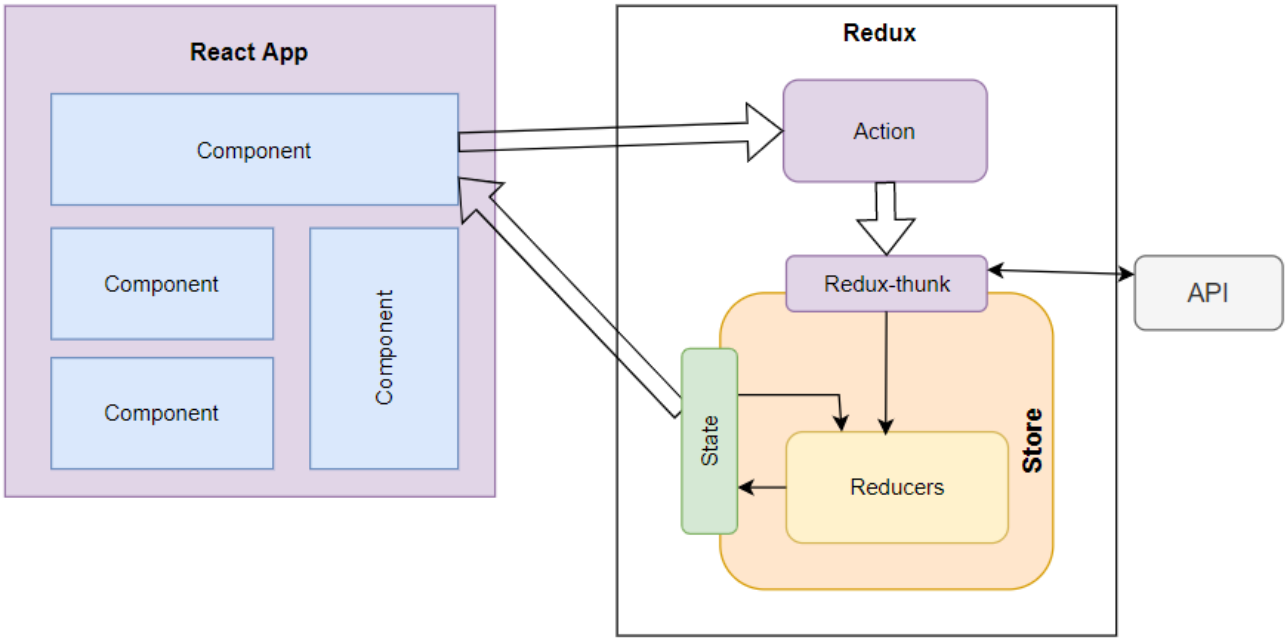


Рисунок 3.5 – React-Redux архітектура

Опис файлів, що були створені для роботи з бібліотекою Redux і містять в собі actions та reducers описані у таблиці 3.3.

Таблиця 3.3 – Опис файлів теки redux

Назва файлу	Опис файлу
dataActions	Містить опис можливих дій, що стосуються оголошень. Такі функції, як getPosts, likePost, deletePost тощо.
userActions	Містить опис можливих дій, що пов’язані з функціоналом, який стосується користувача. Наприклад, loginUser, logoutUser, signupUser, setAuthorizationHeader тощо.
userReducer	Містить код для опрацювання викликаних дій пов’язаних з користувачем. Опрацьовує такі типи дій: SET_USER, SET_AUTHENTICATED, SET_UNAUTHENTICATED, LOADING_USER, LIKE_POST, UNLIKE_POST.
dataReducer	Містить код для опрацювання викликаних дій з файлу dataActions. Змінює частину стану, що стосується оголошень. Як саме має змінитись стан визначається за допомогою типу в казаного в action. Опрацьовує такі типи дій: SET_POSTS, LIKE_POST, UNLIKE_POST, LOADING_DATA, DELETE_POST, ADD_POST, SET_POST, SUBMIT_COMMENT.
uiReducer	Містить код для опрацювання дій пов’язаних з інтерфейсом. Опрацьовує такі типи дій: SET_ERRORS, CLEAR_ERRORS, LOADING_UI, STOP_LOADING_UI
store	Файл, що відповідає за налаштування сховища, приєднує reducers та застосовує проміжне програмне забезпечення redux-thunk.
types	Файл, що містить список допустимих типів дій.

3.3 MaterialUI

Створення візуальних компонентів та їх стилізація в програмному рішенні забезпечується бібліотекою компонентів MaterialUI. Material UI є одною з найбільш популярних бібліотек компонентів для React застосунків. Дана бібліотека надає широкий спектр високоякісних компонентів, що допомагають розробити графічний інтерфейс, що буде приємним для користувача. Крім того, всі компоненти легкі в налаштуванні та стилізації. Бібліотека має обширну та добре описану документацію, що допомагає в розробці.

До елементів, які були використані в даній роботі можна віднести: Typography для розміщення різного роду тексту, Box для об'єднання кількох елементів у групу, Button, App Bar, Card, Link, Avatar тощо.

3.4 Висновки до розділу

В ході роботи над даним розділом було побудовано структуру клієнтської частини застосунку.

Враховуючи функціональні та нефункціональні вимоги до системи було визначено необхідні екрани застосунку (основний екран, екран входу та реєстрації, екран відображення інформації про користувача та про оголошення).

Відповідно до призначення системи та основних правил побудови зручного у користування застосунку було побудовано прототипи, які зображені на рис. 3.1 та 3.2. Визначено основні кольори, шрифти та елементи інтерфейсу.

Було прийнято рішення будувати систему у вигляді односторінкового вебзастосунку. Для такої реалізації використано JavaScript бібліотеку React. Створено необхідні компоненти для відповідності прототипу. Для зручності управління станом системи обрано бібліотеку Redux. Створено необхідні для функціонування системи дії та їх ред'юсери.

В пункті 3.3 описано використану бібліотеку React компонентів Material UI, яку було використано в роботі, для розробки графічного інтерфейсу. Створений застосунок забезпечує необхідний функціонал та приємний досвід користування.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Серед етапів розробки програмного забезпечення дуже важливе місце посідає тестування. Оскільки, навіть якщо успішно пройдено всі попередні етапи створення програмного продукту, це ще не є запорукою успішної роботи системи.

Тестування допомагає не лише оцінити якість роботи системи, але й перевірити відповідність вимогам. Аналіз якості ПЗ, крім того, допомагає зменшити кількість помилок, що виникнуть при користуванні продуктом реальними користувачами.

Оскільки якість програмного забезпечення поняття досить суб’єктивне, то процес тестування може лише дати можливість гарантувати, що всі елементи працюють відповідно до того, як це планувалось при розробці. Наприклад, у відповідь на певний помилковий запит до API відповіддю є помилка тощо.

Аналіз якості програмного забезпечення – це процес, який гарантує, що всі процеси, методи, робочі елементи та дії, що є в проєкті відповідають вимогам та визначеним стандартам. Основною метою тестування є перевірка коректності роботи системи відповідно до визначеного функціонала, в тому числі за різних умов (наприклад, зміни пристрою).

Тестування має враховувати можливі помилки, які виникають, як через неправильне написання програми (наприклад, неправильні функції запиту до БД), так і через помилку в логіці роботи (наприклад, користувач не може перейти на сторінку з детальною інформацією про оголошення з екрана, де розміщені всі його оголошення).

Зазвичай в ході тестування перевіряють такі поняття як коректність, надійність, практичність, безпечність. Хоча цей список може бути більш обширним залежно від типу проєкту і вимог до нього.

В розроблюваній системі потрібно перевірити коректність роботи написаних функцій Firebase та роботу елементів клієнтської частини застосунку.

4.2 Опис процесів тестування

В ході проведення тестування необхідно визначити чи немає помилок в роботі функцій та чи правильно функціонує інтерфейс, з яким взаємодіє користувач.

Перш за все необхідно перевірити коректність відповідей від API залежно від введення вхідних даних чи параметрів. Для цього було використано популярний сьогодні інструмент тестування API Postman. В ході тестування було проведено:

– перевірку відповідності отриманих результатів на відповідність очікуваним;

– перевірку при введенні некоректних даних (пусті поля, некоректний тип даних);

– перевірку на виконання запитів неавторизованим користувачем.

Нижче наведено приклади отриманих відповідей залежно від введених даних:

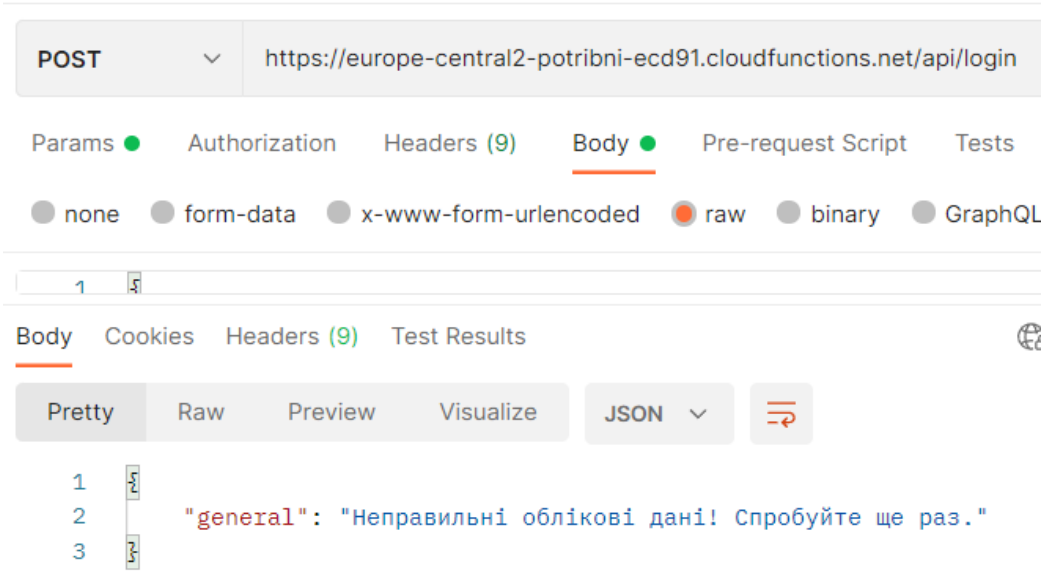


Рисунок 4.1 – Приклад надсилання неправильних даних для входу в систему

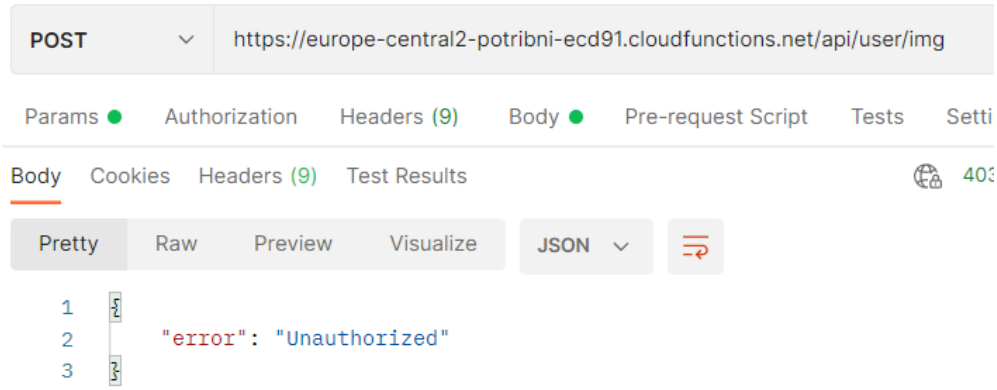


Рисунок 4.2 – Приклад запиту неавторизованим користувачем

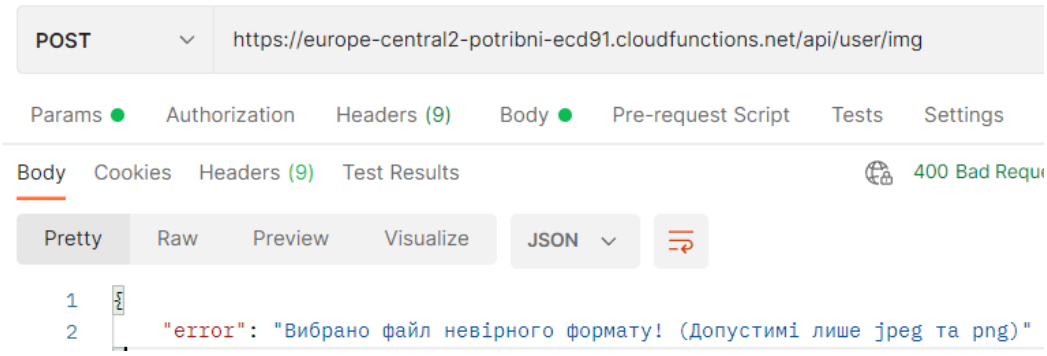


Рисунок 4.3 – Приклад запиту із неправильним форматом вхідних даних

Таким чином було проведено тестування API створеного з допомогою Firebase Functions.

Наступним кроком тестування було тестування функцій застосунку, використовуючи контрольний приклад. До дій у системі, які було протестовано таким чином віднесено:

– пошук необхідних оголошень по категорії та вмісту слів;
– вхід в систему;
– реєстрація в системі;
– додавання нового оголошення;
– збереження оголошення в колекцію вибране;
– коментування оголошення;
– видалення оголошення.

Таблиця 4.1 – Пошук оголошення

Мета тесту	Перевірити роботу функції пошуку
Початковий стан	Користувач увійшов на сайт. Відображаються елементи інтерфейсу, в тому числі панель для пошуку.
Вхідні дані	Текст введений користувачем, категорія (вибрана з меню), місто(вибране з меню), тип оголошення.
Проведення тесту	Натиснути кнопку “Пошук”
Очікуваний результат	На екрані відображаються оголошення, що відповідають пошуковому запиту. Якщо оголошень не знайдено відображається повідомлення: “На жаль, за вашим запитом оголошень не знайдено”.
Фактичний результат	На екрані відобразились оголошення, що відповідають пошуковому запиту. При введенні некоректного запиту, відображається повідомлення: “На жаль, за вашим запитом оголошень не знайдено”.

Таблиця 4.2 – Реєстрація в системі

Мета тесту	Перевірити роботу функції реєстрації в системі, якщо користувач залишив одне з полів пустим
Початковий стан	Користувач не авторизований. Відображається екран для реєстрації в системі із чотирма полями для введення (електронна пошта, ім’я, пароль, підтвердження пароля) та кнопкою “Зареєструватись”.
Вхідні дані	Електронна пошта, пароль, ім’я.

Продовження таблиці 4.2

Проведення тесту	Користувач вводить дані та залишає поле “Підтвердити пароль” пустим. Натискає кнопку “Зареєструватись”
Очікуваний результат	Поле для підтвердження пароля підсвічується червоним і відображається повідомлення про помилку. Користувач не зареєстрований.
Фактичний результат	Поле для підтвердження пароля підсвітилось червоним і відобразилась підказка про помилку. Користувач не зареєстрований.

Таблиця 4.3 – Вхід в систему

Мета тесту	Перевірити роботу функції входу в систему
Початковий стан	Користувач не авторизований. Відображається екран входу в систему з двома полями вводу (електронна пошта та пароль) та кнопкою “Увійти”.
Вхідні дані	Електронна пошта та пароль.
Проведення тесту	Ввести необхідні дані та натиснути на кнопку “Увійти”.
Очікуваний результат	Відображається головний екран застосунку із панеллю для пошуку та списком оголошень. Панель навігації змінює вигляд, зникають кнопки для входу та реєстрації, натомість з’являються кнопки для входу на сторінку користувача, що авторизувався та для перегляду списку “Вибране”.
Фактичний результат	Відобразився головний екран застосунку із панеллю для пошуку та списком оголошень. З панелі навігації зникли кнопки для входу та реєстрації, натомість з’явились кнопки для входу на сторінку користувача, що авторизувався та для перегляду списку “Вибране”.

Таблиця 4.4 – Додавання нового оголошення

Мета тесту	Перевірити функцію для додавання нового оголошення
Початковий стан	Користувач натиснув на кнопку для додавання нового оголошення, перебуваючи на основному екрані. Відображається форма для створення нового оголошення із полями для введення.
Вхідні дані	Назва оголошення (“Віддам дитячу курточку”), категорія (“Речі”), опис (“Віддам весняну курточку на дівчинку 12 років”), місто (“Івано-Франківськ”).
Проведення тесту	Користувач вводить необхідний опис оголошення та натискає кнопку “Створити оголошення”
Очікуваний результат	Відображається анімація завантаження. Форма для створення оголошення зникає. Якщо умови пошуку і параметри оголошення відповідають, то з’являється оголошення.
Фактичний результат	Відобразилась анімація завантаження. Форма для створення оголошення зникла. Параметри оголошення і пошуку відповідають, оголошення з’явилось у списку.

Таблиця 4.5 – Збереження оголошення в колекцію “Вибране”

Мета тесту	Перевірити функцію додавання оголошення в колекцію “Вибране”
Початковий стан	Користувач авторизований, переглядає основний екран зі списком оголошень. На кожному з оголошень є іконка, для додавання його в колекцію “Вибране”. Іконка на обраному оголошенні незафарбована.

Продовження таблиці 4.5

Вхідні дані	Натискання на кнопку для додавання у «Вибране»
Проведення тесту	Користувач натискає на іконку, щоб вподобати оголошення
Очікуваний результат	Іконка змінює вигляд на зафарбований. Оголошення стає доступним у списку “Вибране” даного користувача.
Фактичний результат	Іконка змінила вигляд на зафарбований. Оголошення стало доступним у списку “Вибране” даного користувача.

Таблиця 4.6 – Додавання коментаря до оголошення

Мета тесту	Перевірити функцію додавання коментаря до оголошення
Початковий стан	Користувач переглядає екран з окремим оголошенням. В тому числі з коментарями до оголошення та полем для вводу нового коментаря
Вхідні дані	Текстовий коментар
Проведення тесту	Написати коментар у полі для введення та натиснути кнопку для підтвердження додавання коментаря.
Очікуваний результат	Поле для внесення коментаря очищується, новостворений коментар відображається під оголошенням найвищим.
Фактичний результат	Поле для внесення коментаря очистилось, новостворений коментар відобразився першим у списку коментарів під оголошенням.

Таблиця 4.7 – Видалення оголошення

Мета тесту	Перевірка функції видалення оголошення

Продовження таблиці 4.7

Початковий стан	Відображається екран перегляду інформації про поточного користувача. Відображена інформація про користувача та список оголошень, що розміщений цим користувачем. На кожному з оголошень є кнопка у вигляді іконки для видалення оголошення.
Вхідні дані	Натискання на кнопку для видалення
Проведення тесту	Користувач натискає на кнопку для видалення на одному з оголошень. З'являється вікно із запитанням "Ви впевнені, що бажаєте видалити це оголошення?" Та кнопками-варіантами "Підтвердити" та "Скасувати". Користувач натискає на кнопку "Підтвердити"
Очікуваний результат	Спливаюче вікно зникає. Видалене оголошення більше не відображається у списку.
Фактичний результат	Спливаюче вікно зникло. Видалене оголошення більше не відображалось у списку.

З огляду на інформацію викладену у таблицях, можна зробити висновок, що функції працюють відповідно до зазначених вимог, а тести пройдено успішно.

4.3 Висновки до розділу

В ході роботи над даним розділом було проведено аналіз якості розроблюваної системи. Наведено обгрунтування необхідності тестування. Описано, які основні елементи системи підлягають тестуванню. Тестування потребували функції (Firebase Functions), що формують API, а також елементи інтерфейсу користувача, що забезпечують можливість доступу до основного функціонала системи.

Перевірено відповідність створених функцій до описаних вимог.

Наведено опис процесів тестування. Тестування запитів проводилось з допомогою сервісу Postman. Приклади відповідей на різного типу запити (з помилковими даним, з неповними даними) проілюстровано на рисунках.

Описано тест-кейси для тестування інтерфейсу користувача вебзастосунку у вигляді таблиць. Подано інформацію про очікувані та фактичні результати тестів. Функції, що підлягали тестуванню працюють відповідно до опису та вимог.

5 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Оскільки для реалізації серверної частини було використано Firebase, то для розгортання було прийнято рішення також використовувати один із сервісів – Firebase Hosting.

Firebase Hosting надає певний ряд переваг:

– хостинг гарантує швидку доставку вмісту до користувача, оскільки кожен завантажений файл знаходиться на SSD;

– швидке розгортання;

– у випадку, якщо при розгортанні було допущено помилку, зробити “відкат” можна в один клік.

З допомогою Firebase CLI розгортаються файли, що містяться в локальному сховищі на сервері Firebase Hosting. Сервер має конфігурацію SSL у глобальній CDN, що забезпечує безпеку та стабільність зв’язку.

Для того, щоб розгорнути створений застосунок з використанням Firebase Hosting потрібно виконати наступні кроки:

– в терміналі середовища VS Code або в командному рядку перебуваючи в директорії, що містить всі файли застосунку виконати команду “npm run build”;

– у консолі Firebase перейти до меню створюваного проєкту та активувати сервіс Hosting

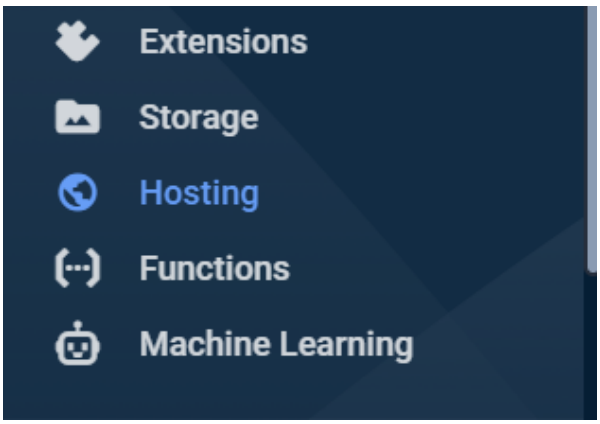


Рисунок 5.1 – Вкладка хостингу в меню проєкту

– у термінал VS Code виконати команду “npm install -g firebase-tools”;

– далі виконати команду “firebase init” тим самим проініціалізувавши проєкт firebase у поточній директорії;

– провести подальше налаштування відповідно до інструкцій, що з’являтимуться в рядку;

– в ролі папки, файли якої будуть завантажені до хостингу, вибрати папку build;

– в терміналі виконати команду “firebase deploy”.

В результаті виконання цих кроків у терміналі з’явиться посилання для доступу до створеного вебзастосунку.

Firebase Hosting підтримує безпечний протокол з’єднання HTTPS та надає й автоматично налаштовує сертифікат SSL для розгорнутого застосунку. Протокол дозволяє захистити передачу даних від атак типу: розкриття шифрів, Man-in-the-Middle, атака відгуку тощо.

5.2 Робота з програмним забезпеченням

Для того, щоб скористатись створеним вебзастосунком необхідно перейти за посиланням отриманим в результаті розгортання програмного забезпечення. Детальніше роботу з системою описано в документі “Керівництво користувача”.

5.3 Висновки до розділу

В ході роботи над даним розділом було описано процес розгортання вебзастосунку з допомогою сервісу Firebase Hosting. Надано обгрунтування саме такого способу розгортання.

Надано коротку інформацію про роботу із застосунком. Детальнішу інформацію викладено в документі “Керівництво користувача”.

ВИСНОВКИ

В ході роботи над даним дипломним проєктом було розглянуто етапи створення системи для обміну пропозиціями для допомоги внутрішньо переміщеним особам.

В першому етапі роботи над застосунком було проаналізовано предметну область. В процесі аналізу було визначено, що на даний час в Україні є чимало ВПО, які потребують не лише фінансової, але й різного роду гуманітарної допомоги. Водночас є велика кількість громадян , що виявляють бажання зробити свій вклад в підтримку співвітчизників. Такі дані доводять актуальність створюваного програмного рішення.

Крім того, в першому розділі було сформульовано набір функціональних та нефункціональних вимог, визначено цілі та задачі розробки, основними з яких є створення зручного інтерфейсу з можливістю додавання та пошуку оголошень.

У процесі роботи над другим розділом було описано архітектуру серверної частини застосунку. Для реалізації системи було обрано архітектурний підхід Serverless, за умови використання якого зменшуються витрати (в тому числі й часові) на реалізацію та підтримку серверної інфраструктури. В ролі BaaS було обрано платформу Firebase, що надає сервіси для хмарного зберігання даних, аутентифікації користувачів, хостингу, FaaS та інші.

У якості середовища розробки було обрано Visual Studio Code.

Було створено набір функцій для роботи з базою даних Firestore та Storage, які покрили необхідний функціонал, визначений у вимогах.

Наступним кроком та відповідно наступним розділом було створення клієнтської частини застосунку. Було прийнято рішення реалізовувати застосунок у вигляді Single Page Application. Для реалізації такого рішення було обрано бібліотеку React, а для управляння станами бібліотеку Redux. Було створено прототипи та розроблено графічний інтерфейс користувача.

У четвертому та п’ятому розділах описано процес тестування та розгортання. Тестування серверних функцій було проведено з використанням інструменту Postman. Функціонування інтерфейсу проведено шляхом ручного тестування:

перевірено роботу основних функціональних елементів.

Розгортання вебзастосунку було проведено з допомогою сервісу Firebase Hosting.

В результаті роботи над застосунком було створено систему, що дозволяє користувачам авторизуватися, створювати оголошення, здійснювати пошук серед них, зберігати їх, додаючи у колекцію “Вибране”, видаляти їх.

Серед функціональних вимог до системи була вказана можливість комунікувати з іншими користувачами. Її реалізовано у вигляді можливості додавати коментарі до оголошень.

Створений застосунок є простим у користуванні та надає можливість простого обміну пропозиціями допомоги. У майбутньому систему можна розвивати, як засіб для підтримки не тільки ВПО, а й будь-якого кола людей, що потребує різного роду гуманітарної допомоги. Можна розширити функціонал, наприклад, додавши можливість надсилати короткі повідомлення іншим користувачам. Також можна долучити волонтерські чи благодійні організації до роботи із застосунком, щоб запити про потребу опрацьовувались більшим колом осіб.

Таким чином, дещо допрацювавши застосунок, можна продовжити його використання навіть після того, як війна закінчиться нашою перемогою і допомога ВПО стане менш актуальною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Звіт про внутрішнє переміщення в Україні. Опитування загального населення Раунд 4 3 травня 2022 року, 2022. URL: https://displacement.iom.int/reports/zvit-pro-vnutrishne-peremischennya-v-ukraini-opituvannya-zagalnogo-naselennya-raund-4-3 (дата звернення: 05.05.2022).

2) Динаміка реєстрацій благодійних організацій під час війни: дослідження Vkursi та Zagoriy Foundation, 2022. URL: https://vkursi.pro/news/content/splesk-reiestratsiy-blahodiynykh-orhanizatsiy-doslidzhennia-vkursi-ta-zagoriy-foundation-234986 (дата звернення: 10.05.2022).

3) Rajan R.A.P. Serverless Architecture - A Revolution in Cloud Computing. *Tenth International Conference on Advanced Computing (ICoAC).* 2018. P.88-93. DOI: 10.1109/ICoAC44903.2018.8939081.

4) Jason Ma. Serverless Architectures: The Evolution of Cloud Computing. 2017. URL: https://www.mongodb.com/blog/post/serverless-architectures-the-evolution-of-cloud-computing (дата звернення: 10.05.2022)

5) Посібник: знайомство з React. URL: https://uk.reactjs.org/tutorial/tutorial.html#what-is-react (дата звернення 07.05.2022)

6) Lynn T., Rosati P., Lejeune A., Emeakaroha V. A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. *IEEE International Conference on Cloud Computing Technology and Science (CloudCom).* 2017. P. 162-169. DOI: 10.1109/CloudCom.2017.15.

7) Firebase Authentication. URL: https://firebase.google.com/docs/auth (дата звернення: 10.05.2022)

8) Firebase Authentication Password Hashing. URL: https://firebase opensource .com /projects /firebase/scrypt/ (дата звернення: 10.05.2022)

9) Ertaul L., Kaur M., Gudise V. A. K. R.. Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms. *In Proceedings of the International Conference on Wireless Networks (ICWN)*. Athens, 2016. P. 66–72. URL: http://mcs.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf

10) Security Measures Algolia. URL: https://www.algolia.com/distributed - secure/security-compliance/measures/ (дата звернення: 15.05.2022)

11) Nielsen J. Usability 101: Introduction to Usability. 2012. URL: https:// www.nngroup.com/articles/usability-101-introduction-to-usability/ (дата звернення: 15.05.2022)

12) Solovei V., Olshevska O., Bortsova, Y. THE DIFFERENCE BETWEEN DEVELOPING SINGLE PAGE APPLICATION AND TRADITIONAL WEB APPLICATION BASED ON MECHATRONICS ROBOT LABORATORY ONAFT APPLICATION. *Automation of technological and business processes.* 2018. Vol.10, № 1. P. 6 DOI: https://doi.org/10.15673/atbp.v10i1.874.

13) React Documentation. URL: https://uk.reactjs.org/ (дата звернення 15.05.2022)

14) Three Principles Redux. URL: https://redux.js.org/understanding/thinking - in-redux/three-principles. (дата звернення 15.05.2022)

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ВЕБ-ЗАСТОСУНОК ОБМІНУ ПРОПОЗИЦІЯМИ ДЛЯ ДОПОМОГИ ВНУТРІШНЬО ПЕРЕМІЩЕНИМ ОСОБАМ

Опис програми

КПІ.ІТ-8216.045440.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена СИРОТА

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Мар’яна ЛОЗЯК

Київ – 2022

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам

(Найменування програми (документа))

CD-R

(Вид носія даних)

22 арк, 755700 Кб

(Обсяг програми, арк., Кб)

Київ – 2022

handlers/post.js

```
const { db } = require('../util/admin');
const algoliasearch = require("algoliasearch");
const algolia = algoliasearch();

exports.search = (req, res) => {
  const query = req.query.searchText;
  const category = req.query.category;
  const place = req.query.place;
  const offer = req.query.isOffer;
  const client = algoliasearch('9QY4355JTC',
    '8f85921ac1f8ef1d074f9d08029b1b44');
  const index = client.initIndex('posts');

  let filter;
  if ((category === '"""')&&(place==='"""')) {
     filter = `isOffer: ${offer}`;
  } else if (category === '"""){
      filter = `place: ${place} AND isOffer: ${offer}`;
  } else if( place === '"""){
    filter = `category: ${category} AND isOffer: ${offer}`;
  } else {
    filter = `category: ${category} AND place: ${place}  AND isOffer: ${offer}`;
  }
    index
    .search(query, {
      filters: filter
    })
    .then(({ hits }) => {
      console.log(hits);
      let posts = [];
      hits.forEach(doc => {
        posts.push({
          postId: doc.objectID,
          userHandle: doc.userHandle,
          userId: doc.userId,
          title: doc.title,
          category: doc.category,
          place: doc.place,
          body: doc.body,
          createdAt: doc.createdAt,
          likeCount: doc.likeCount,
          isOffer: doc.isOffer
        });
      });
      res.json(posts);
    }).catch(err => console.error(err));
}

function pushPosts(data) {
  let posts = [];
```

```
  data.forEach(doc => {
    posts.push({
      postId: doc.id,
      userHandle: doc.data().userHandle,
      userId: doc.data().userId,
      title: doc.data().title,
      category: doc.data().category,
      place: doc.data().place,
      body: doc.data().body,
      createdAt: doc.data().createdAt,
      likeCount: doc.data().likeCount,
      isOffer: doc.data().isOffer
    });
  });
  return posts;
}

exports.getAllPosts = (req, res) => {


  const limit = req.query.limit;
  if (req.query.last) {
    const last = req.query.last;
    db.collection("posts")
      .orderBy('createdAt', 'desc')
      .startAfter(last)
      .limit(parseInt(limit))
      .get()
      .then(data => {
        let posts = [];
        posts = pushPosts(data);
        const lastKey = posts[posts.length - 1].createdAt;
        return res.json({ posts, lastKey });
      })
      .catch(err => console.error(err));
  } else {

    db.collection("posts")
      .orderBy('createdAt', 'desc')
      .limit(parseInt(limit))
      .get()
      .then(data => {
        let posts = [];
        posts = pushPosts(data);
        const lastKey = posts[posts.length - 1].createdAt;
        return res.json({ posts, lastKey });
      })
      .catch(err => console.error(err));
  }
};

exports.getAllOffers = (req, res) => {
  db.collection('posts')
```

```
    .where('isOffer', '==', true)
    .orderBy('createdAt', 'desc')
    .get()
    .then(data => {
      let posts = [];
      posts = pushPosts(data);
      res.json(posts);
    })
    .catch(err => console.error(err));
}

exports.getAllNeeds = (req, res) => {
  db.collection('posts')
    .where('isOffer', '==', false)
    .orderBy('createdAt', 'desc')
    .get()
    .then(data => {
      let posts = [];
      posts = pushPosts(data);
      res.json(posts);
    })
    .catch(err => console.error(err));
}



exports.postOne = (req, res) => {
  const newPost = {
    userHandle: req.user.handle,
    userId: req.user.id,
    title: req.body.title,
    category: req.body.category,
    place: req.body.place,
    body: req.body.body,
    createdAt: new Date().toISOString(),
    likeCount: 0,
    isOffer: req.body.isOffer
  };


  db
    .collection('posts')
    .add(newPost)
    .then(doc => {
      const resPost = newPost;
      resPost.postId = doc.id;
      res.json(resPost);
    })
    .catch(err => {
      res.status(500).json({ error: 'Щось пішло не так!' });
      console.error(err);
    });
};
```

```
exports.getOnePost = (req, res) => {
  let postData = {};
  db
    .doc(`/posts/${req.params.postId}`)
    .get()
    .then((doc) => {
      if (!doc.exists) {
        return res.status(404).json({ error: 'Такого оголошення не знайдено!' })
      }
      postData = doc.data();
      postData.postId = doc.id;
      return db.collection('comments')
        .orderBy('createdAt', 'desc')
        .where('postId', '==', req.params.postId)
        .get();

    })
    .then((data) => {
      postData.comments = [];
      data.forEach(doc => {
        postData.comments.push(doc.data())
      });
      return res.json(postData);
    })
    .catch(err => {
      console.error(err);
      return res.status(500).json({ error: err.code });
    });
};


exports.likePost = (req, res) => {
  const likeDocument = db
    .collection('favourites')
    .where('userId', '==', req.user.id)
    .where('postId', '==', req.params.postId)
    .limit(1);

  const postDocument = db.doc(`/posts/${req.params.postId}`);

  let postData;

  postDocument
    .get()
    .then((doc) => {
      if (doc.exists) {
        postData = doc.data();
        postData.postId = doc.id;
        return likeDocument.get();
      } else {
        return res.status(404).json({ error: 'Оголошення не знайдено!' });
      }
```

```
    })
    .then((data) => {
      if (data.empty) {
        return db
          .collection('favourites')
          .add({
            postId: req.params.postId,
            userId: req.user.id,
            postTitle: postData.title,
            createdAt: new Date().toISOString()
          })
          .then(() => {
            postData.likeCount++;
            return postDocument.update({ likeCount: postData.likeCount });
          })
          .then(() => {
            return res.json(postData);
          });
      } else {
        return res.status(400).json({ error: 'Оголошення вже додано до Вибраних' });
      }
    })
    .catch((err) => {
      console.error(err);
      res.status(500).json({ error: err.code });
    });
};

exports.unlikePost = (req, res) => {
  const likeDocument = db
    .collection('favourites')
    .where('userId', '==', req.user.id)
    .where('postId', '==', req.params.postId)
    .limit(1);

  const postDocument = db.doc(`/posts/${req.params.postId}`);

  let postData;

  postDocument
    .get()
    .then((doc) => {
      if (doc.exists) {
        postData = doc.data();
        postData.postId = doc.id;
        return likeDocument.get();
      } else {
        return res.status(404).json({ error: 'Оголошення не знайдено!' });
      }
    })
    .then((data) => {
      if (data.empty) {
        return res.status(400).json({ error: 'Оголошення не було додано до Вибраних' });
```

```
      } else {
        return db
          .doc(`/favourites/${data.docs[0].id}`)
          .delete()
          .then(() => {
            postData.likeCount--;
            return postDocument.update({ likeCount: postData.likeCount });
          })
          .then(() => {
            res.json(postData);
          });
      }
    })
    .catch((err) => {
      console.error(err);
      res.status(500).json({ error: err.code });
    });
};

exports.commentOnPost = (req, res) => {
  if (req.body.body.trim() === '')
    return res.status(400).json({ comment: 'Поле не може бути пустим!' });

  const newComment = {
    body: req.body.body,
    createdAt: new Date().toISOString(),
    postId: req.params.postId,
    userId: req.user.id,
    userHandle: req.user.handle,
    userImage: req.user.imageUrl
  };
  console.log(newComment);

  db.doc(`/posts/${req.params.postId}`)
    .get()
    .then((doc) => {
      if (!doc.exists) {
        return res.status(404).json({ error: 'Оголошення не знайдено!' });
      }
    })
    .then(() => {
      return db.collection('comments').add(newComment);
    })
    .then(() => {
      res.json(newComment);
    })
    .catch((err) => {
      console.log(err);
      res.status(500).json({ error: 'Щось пішло не так!' });
    });
}

exports.deletePost = (req, res) => {
```

```
  const document = db.doc(`/posts/${req.params.postId}`);
  document
    .get()
    .then((doc) => {
      if (!doc.exists) {
        return res.status(404).json({ error: 'Оголошення не знайдено' });
      }
      if (doc.data().userHandle !== req.user.handle) {
        return res.status(403).json({ error: 'Ви не авторизовані!' });
      } else {
        return document.delete();
      }
    })
    .then(() => {
      res.json({ message: 'Оголошення видалено успішно!' });
    })
    .catch((err) => {
      console.error(err);
      return res.status(500).json({ error: err.code });
    });
};
```

handlers/user.js

```
const { admin, db } = require("../util/admin");
const { v4 : uuidv4 } = require('uuid');
const {
    validateSignupData,
    validateLoginData,
    reduceUserDetails
} = require("../util/validators");
const firebaseConfig = require("../util/config");
const firebase = require('firebase');

firebase.initializeApp(firebaseConfig);
exports.signup = (req, res) => {
    const newUser = {
        email: req.body.email,
        password: req.body.password,
        confirmPassword: req.body.confirmPassword,
        handle: req.body.handle,
    };
  const { valid, errors } = validateSignupData(newUser);

  if (!valid) return res.status(400).json(errors);

  const defaultImg = 'user_icon.png';

  let token, userUID;
   firebase.auth()
   .createUserWithEmailAndPassword(newUser.email, newUser.password)
    .then((data) => {
            userUID = data.user.uid;
```

```
        return data.user.getIdToken();
      })
    .then((idToken) => {
      token = idToken;
      const userCredentials = {
        handle: newUser.handle,
        email: newUser.email,
        createdAt: new Date().toISOString(),
        imageUrl:
`https://firebasestorage.googleapis.com/v0/b/${firebaseConfig.storageBucket}/o/${defaultImg}?alt=media&to
ken=6c9c7c3e-ed3e-41db-9fd6-8527447967a4`,
        userUID
        };
      return db.collection('users').add(userCredentials);
    })
    .then(() => {
     return res.status(201).json({token});
    })
    .catch((err) => {
     console.error(err);
        if (err.code === "auth/email-already-in-use") {
          return res
            .status(400)
            .json({ email: "Акаунт з такою поштою вже існує!" });
        } else {
      return res.status(500).json({general: 'Щось пішло не так! Будь ласка, спробуйте ще раз.'});
     }
    });
};
exports.login = (req, res) => {
    const user = {
      email: req.body.email,
      password: req.body.password,
    };

    const { valid, errors } = validateLoginData(user);

    if (!valid) return res.status(400).json(errors);

    firebase
      .auth()
      .signInWithEmailAndPassword(user.email, user.password)
      .then((data) => {
        return data.user.getIdToken();
      })
      .then((token) => {
        return res.json({ token });
      })
      .catch((err) => {
        console.error(err);
        return res
              .status(403)
              .json({ general: "Неправильні облікові дані! Спробуйте ще раз." });
```

```
      });
};

exports.uploadImage = (req, res) => {
   const BusBoy = require("busboy");
   const path = require("path");
   const os = require("os");
   const fs = require("fs");
   const busboy = BusBoy({ headers: req.headers });
   let imageFileName;
   let imageToBeUploaded = {};

   let generatedToken = uuidv4();
   busboy.on("file", (fileName, file, info) => {

      const { filename, encoding, mimeType } = info;
      console.log(filename);
      if (mimeType !== "image/jpeg" && mimeType !== 'image/png') {
         return res.status(400).json({ error: "Вибрано файл невірного формату! (Допустимі лише jpeg та png)"
});
      }
      const imageExtension = filename.split(".")[
         filename.split(".").length - 1
      ];
      console.log(imageExtension);
      imageFileName = `${Math.round(
         Math.random() * 100000000000
      ).toString()}.${imageExtension}`;
      const filepath = path.join(os.tmpdir(), imageFileName);
      imageToBeUploaded = { filepath, mimeType };
      file.pipe(fs.createWriteStream(filepath));
   });
   busboy.on("close", () => {
      admin
         .storage()
         .bucket()
         .upload(imageToBeUploaded.filepath, {
            resumable: false,
            metadata: {
               metadata: {
                  contentType: imageToBeUploaded.mimetype,
                  firebaseStorageDownloadTokens: generatedToken
               },
            },
         })
         .then(() => {
            const imageUrl =
`https://firebasestorage.googleapis.com/v0/b/${firebaseConfig.storageBucket}/o/${imageFileName}?alt=medi
a&token=${generatedToken}`;

            return db.doc(`/users/${req.user.id}`).update({ imageUrl });
         })
         .then(() => {
```

```
            return res.json({ message: "Зображення завантажено успішно" });
          })
          .catch((err) => {
            console.error(err);
            return res.status(500).json({ error: err.code });
          });
    });
    busboy.end(req.rawBody);
};
exports.getUserFavourites = (req,res) => {
    db.collection('favourites')
      .where('userId','==',req.user.id)
      .orderBy('createdAt','desc')
      .get()
      .then( data =>{
        let favourites=[];
        data.forEach((doc)=>{
          favourites.push(doc.data());
        });
        res.json(favourites);
      })
      .catch(err => console.error(err));
}
exports.addUserDetails = (req, res) => {
    let {userDetails, error} = reduceUserDetails(req.body);
    if( Object.keys(error).length !== 0){
      return res.status(400).json(error);
    }

    db.doc(`/users/${req.user.id}`)
      .update(userDetails)
      .then(() => {
        return res.json({ message: "Профіль оновлено успішно" });
      })
      .catch((err) => {
        console.rror(err);
        return res.status(500).json({ errror: errcode });
      });
};
exports.getUserDetails = (req, res) => {
    let userData = {};
    db.doc(`/users/${req.params.userId}`)
      .get()
      .then((doc) => {
        if (doc.exists) {

          userData.user = doc.data();
          const result =  db
            .collection("posts")
            .where("userId", "==", req.params.userId)
            .get();
          console.log(result);
              return result;
```

```
        } else {
          return res.status(404).json({ error: "Користувача не знайдено!" });
        }
      })
      .then((data) => {
        console.log(data);
        userData.posts = [];
        data.forEach((doc) => {
          userData.posts.push({
            title: doc.data().title,
            category: doc.data().category,
            place: doc.data().place,
            body: doc.data().body,
            createdAt: doc.data().createdAt,
            userHandle: doc.data().userHandle,
            userId: doc.data().userId,
            likeCount: doc.data().likeCount,
            isOffer: doc.data().isOffer,
            postId: doc.id
          });
        });
        return res.json(userData);
      })
      .catch((err) => {
        console.error(err);
        return res.status(500).json({ error: err.code });
      });
};
exports.getCurrentUser = (req, res) => {
  let userData = {};
  db.doc(`/users/${req.user.id}`)
    .get()
    .then((doc) => {
      if (doc.exists) {
        userData.credentials = {userId: doc.id, ...doc.data()};
        return db
          .collection("favourites")
          .where("userId", "==", req.user.id)
          .get();
      }
    })
    .then((data) => {
      userData.favourites = [];
      data.forEach((doc) => {
        userData.favourites.push(doc.data());
      });
      return res.json(userData);
    .catch((err) => {
      console.error(err);
      return res.status(500).json({ error: err.code });
    });
};
```

App.js

```
import React, { Component } from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import './App.css';
//COmponents
import Navbar from './components/Navbar';
import AuthRoute from './utils/AuthRoute';

//Pages
import Home from './pages/Home'
import Login from './pages/Login.js';
import Signup from './pages/Signup';
import PostDialog from './pages/PostDialog';
import Favourites from './pages/Favourites';
import User from './pages/User';
import CurrentUser from './pages/CurrentUser';
//Redux
import { Provider } from 'react-redux';
import store from './redux/store';
import { SET_AUTHENTICATED } from './redux/types';
import { logoutUser, getUserData } from './redux/actions/userActions';
import axios from 'axios';
import { ThemeProvider, createTheme } from '@mui/material/styles';
import themeFile from './utils/theme.js';
import jwtDecode from 'jwt-decode';

const theme = createTheme(themeFile)
const token = localStorage.AuthToken;
if (token) {
  const decodedToken = jwtDecode(token);
  if (decodedToken.exp * 1000 < Date.now()) {
    store.dispatch(logoutUser());
    window.location.href = '/login'
  } else {
    store.dispatch({ type: SET_AUTHENTICATED });
    axios.defaults.headers.common['Authorization'] = token;
    store.dispatch(getUserData());
  }
}
class App extends Component {
  render() {
    return (
      <ThemeProvider theme={theme}>
        <Provider store={store}>
          <Router>
            <Navbar />
            <div className="container" >
              <Routes>
                <Route path="/" element={<Home />} exact />
                <Route exact path='/login' element={<AuthRoute Component={Login} />} />
```

```
            <Route exact path="/signup" element={<AuthRoute Component={Signup} />} />
            <Route exact path="/posts/:postId" element={<PostDialog />} />
            <Route exact path="/user/favourites" element={<Favourites />} />
            <Route exact path="/user/:userId" element={<User />} />
            <Route exact path="/user" element={<CurrentUser />} />
          </Routes>
        </div>
      </Router>
    </Provider>
  </ThemeProvider>);};}

export default App;
```

Home.js

```
import React, { useEffect, useState } from 'react';
import Grid from '@mui/material/Grid';
import axios from 'axios';
import SearchBar from '../components/SearchBar';
import algoliasearch from 'algoliasearch/lite';
import PostBlock from '../components/PostBlock';
import SegmentedView from '../components/SegmentedView';
import { useSelector, useDispatch } from 'react-redux';
import { getOffers } from '../redux/actions/dataActions';

const searchClient = algoliasearch('9QY4355JTC', '8f85921ac1f8ef1d074f9d08029b1b44');
const Home = () => {
 const dispatch = useDispatch();
 const [posts, setPosts] = useState([]);
 const postsGlobal = useSelector(state => state.data.posts);
 useEffect(() => {

  dispatch(getOffers());
 }, []);
 useEffect(() => {
  setPosts(postsGlobal);
 }, [postsGlobal]);

 return (
  <Grid container
   spacing={1}
   direction="column"
   justifyContent="center"
  >
   <Grid item sx={{
   }}>
    <SegmentedView />
   </Grid>

   <Grid item sx={{
    marginBottom: '26px'
   }}>
```

```
      <SearchBar />
    </Grid>
    <Grid item
      sx={{
        width: '100%',
        margin: 'auto'
      }}>
      <PostBlock posts={posts} />

    </Grid>
  </Grid>
 )
}
```

export default Home;

Navbar.js

```
import React, { Fragment } from 'react'
import { Link } from 'react-router-dom';
//mui
import AppBar from '@mui/material/AppBar';
import Box from "@mui/material/Box";
import Toolbar from "@mui/material/Toolbar";
import IconButton from "@mui/material/IconButton";
import Typography from "@mui/material/Typography";
import Menu from "@mui/material/Menu";
import Container from "@mui/material/Container";
import Avatar from "@mui/material/Avatar";
import Tooltip from "@mui/material/Tooltip";
import MenuItem from "@mui/material/MenuItem";

//icon
import ListItemIcon from '@mui/material/ListItemIcon';
import HowToRegRoundedIcon from '@mui/icons-material/HowToRegRounded';
import LoginRoundedIcon from '@mui/icons-material/LoginRounded';
import LogoutRoundedIcon from '@mui/icons-material/LogoutRounded';
import BookmarkRoundedIcon from '@mui/icons-material/BookmarkRounded';
import { BOOKMARK_ON } from '../utils/images'
import { AccountCircle } from '@mui/icons-material';
import { useSelector, useDispatch } from 'react-redux';
import { logoutUser } from '../redux/actions/userActions';
import AddPost from './AddPost';

function Navbar() {

  const dispatch = useDispatch();
  const authenticated = useSelector(state => state.user.authenticated);
  const user = useSelector(state => state.user.credentials);
  const [anchorElUser, setAnchorElUser] = React.useState(null);
  const handleOpenUserMenu = (event) => {
    setAnchorElUser(event.currentTarget);
```

```
  };
  const handleCloseUserMenu = () => {
   setAnchorElUser(null);
  };
  return (
   <AppBar position="static" style={{ background: 'transparent', boxShadow: 'none' }}>

     <Container maxWidth="xl" >
      <Toolbar className="nav-container" disableGutters sx={{ display: "flex", justifyContent: 'space-between'
}} >
        <Link to='/' textalign="center">
         <Avatar alt="Potribni" src={`${process.env.PUBLIC_URL}/ic.png`} variant="square" />
        </Link>
        <Typography
         variant="h5"
         noWrap
         component="a"
         href="/"
         sx={{
          mr: 2,
          display: { xs: "none", md: "flex" },
          flexGrow: 1,
          fontWeight: '900',
          letterSpacing: ".2rem",
          color: "secondary.main",
          textDecoration: "none",
          margin: 4
         }}
        >
         POTRIBNI
        </Typography>

        {authenticated ? (
         <Fragment>
          <AddPost marginRight='15px' />
          <Box sx={{ flexGrow: 0, display: { xs: "none", md: "flex" }, gap: '15px', alignItems: "center", }}>
           <Link to='/user'>
            <Typography
             color='secondary'
             sx={{
              fontSize: "17px",
              fontWeight: '700'
             }}>{user.handle}</Typography></Link>
           <Link to='/user/favourites'>
            <BOOKMARK_ON />
           </Link>

           <Box onClick={() => {
            dispatch(logoutUser());
           }}>
            <LogoutRoundedIcon color='secondary' fontSize='large'
            />
           </Box>
```

```
</Box>

<Box sx={{ flexGrow: 0, display: { xs: "flex", md: "none" } }}>
 <Tooltip title="Меню користувача">
  <IconButton onClick={handleOpenUserMenu} sx={{ p: 0 }}>
   <Avatar alt="Аватар" src={`${process.env.PUBLIC_URL}/user.png`} />
  </IconButton>
 </Tooltip>

 <Menu
  className="menu"
  sx={{
   mt: "45px",
   fontFamily: "monospace",
  }}
  id="menu-appbar"
  anchorEl={anchorElUser}
  anchorOrigin={{
   vertical: "top",
   horizontal: "center"
  }}
  keepMounted
  transformOrigin={{
   vertical: "top",
   horizontal: "center"
  }}
  open={Boolean(anchorElUser)}
  onClose={handleCloseUserMenu}
 >
  <Link to='/user' textalign="center">
   <MenuItem
    onClick={handleCloseUserMenu}>
    <ListItemIcon>
     <AccountCircle fontSize="small" />
    </ListItemIcon>
    Профіль
   </MenuItem>
  </Link>
  <Link to='/user/favourites' texalign="center">
   <MenuItem onClick={handleCloseUserMenu}>
    <ListItemIcon>
     <BookmarkRoundedIcon fontSize='small' />
    </ListItemIcon>
    Вибране
   </MenuItem>
  </Link>

  <Link to='/signup' texalign="center">
   <MenuItem onClick={handleCloseUserMenu}>
    <ListItemIcon>
     <LogoutRoundedIcon fontSize='small' />
    </ListItemIcon>
    Вийти
```

```
          </MenuItem>
        </Link>
      </Menu>
    </Box>
  </Fragment>

) : (
  <Fragment>
    <Box sx={{ flexGrow: 0, display: { xs: "none", md: "flex" }, gap: '15px' }}>
      <Link to='/login'>
        <Typography
          color='secondary'
          sx={{
            fontSize: "17px",
            fontWeight: '700'
          }}>Увійти</Typography></Link>
      <Link to='/signup'><Typography color='secondary'
        sx={{
          fontSize: "17px",
          fontWeight: '700'
        }}>Зареєструватись</Typography></Link>
    </Box>

    <Box sx={{ flexGrow: 0, display: { xs: "flex", md: "none" } }}>
      <Tooltip title="Меню користувача">
        <IconButton onClick={handleOpenUserMenu} sx={{ p: 0 }}>
          <Avatar alt="Аватар" src={`${process.env.PUBLIC_URL}/user.png`} />
        </IconButton>
      </Tooltip>
      <Menu
        className="menu"
        sx={{
          mt: "45px",
          fontFamily: "monospace",
        }}
        id="menu-appbar"
        anchorEl={anchorElUser}
        anchorOrigin={{
          vertical: "top",
          horizontal: "center"
        }}
        keepMounted
        transformOrigin={{
          vertical: "top",
          horizontal: "center"
        }}
        open={Boolean(anchorElUser)}
        onClose={handleCloseUserMenu}
      >
        <Link to='/login' textalign="center">
          <MenuItem
            onClick={handleCloseUserMenu}>
            <ListItemIcon>
```

```
                <LoginRoundedIcon fontSize="small" />
              </ListItemIcon>
              Увійти
            </MenuItem>
          </Link>

          <Link to='/signup' texalign="center">
            <MenuItem onClick={handleCloseUserMenu}>
              <ListItemIcon>
                <HowToRegRoundedIcon fontSize='small' />
              </ListItemIcon>
              Зареєструватись
            </MenuItem>
          </Link>
        </Menu>
      </Box>
      <AddPost />
    </Fragment>
   )}
  </Toolbar>
 </Container>
</AppBar>
);
}

export default Navbar
```

Post.js

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';

//mui
import Box from '@mui/material/Box';
import Card from '@mui/material/Card';
import CardContent from '@mui/material/CardContent';
import Typography from '@mui/material/Typography';
import Chip from '@mui/material/Chip'

import { useSelector } from 'react-redux';
import { categoryColors } from "../utils/categoryColors";
import { LOCATION_SVG } from '../utils/images';
import BookmarkButton from './BookmarkButton';
import DeleteButton from './DeleteButton';

function Post(props) {
 const usId = useSelector((state) => state.user.credentials.userId);
 const {
  post: {
   body,
   userId,
   title,
```

```
    postId,
    category,
    place,
    createdAt
  }
} = props;
return (

 <Card
  sx={{
   background: '#fff',
   borderRadius: '28px',
  }
  }>
  <CardContent
   sx={{
    padding: '30px',
   }}>
   <Box
    component={Link}
    to={`/posts/${postId}`}
    sx={{
     display: 'flex',
     justifyContent: 'space-between',
     marginBottom: "6px",
    }}
   >
    <Typography component="div"
     color="#262522"
     sx={{
      fontSize: '3vh',//26px
      fontWeight: '700',
      lineHeight: '1',
      opacity: '1'
     }}
    >
     {title}
    </Typography>
    <Chip label={category}
     sx={{
      padding: '8px',
      borderRadius: '10px',
      fontSize: '2.3vh',//20px
      backgroundColor: `rgba(${categoryColors[category]}, 0.15)`,
      color: `rgba(${categoryColors[category]}, 1)`
     }} />
   </Box>

   <Typography
    sx={{
     marginBottom: '30px',
     color: '#929290',
    }}>
```

```
        {new Date(createdAt).toLocaleDateString('uk-UA',
         {
          year: 'numeric',
          month: 'long',
          day: 'numeric'
         }
        )}
       </Typography>
       <Typography
        sx={{
         marginBottom: '24px',
         fontSize: '2.4vh',//22px
         fontWeight: '400',
         color: '#3C3B39',
         lineHeight: '1',
         overflow: "hidden",
         textOverflow: "ellipsis",
         display: "-webkit-box",
         WebkitLineClamp: "3",
         WebkitBoxOrient: "vertical",

        }}>
        {body}
       </Typography>
       <Box
        sx={{
         display: 'flex',
         justifyContent: 'space-between',
        }}
       >
        <Typography
         sx={{
          display: "flex",
          alignItems: "center",
          gap: '12px',
          fontSize: '18px',
          color: '#929290'
         }} >
         <LOCATION_SVG />
         {place}
        </Typography>
        <Box sx={{
         display: 'flex'
        }}>
         {(userId === usId) ? (<DeleteButton postId={postId} />) : (<BookmarkButton postId={postId} />)}
        </Box>
       </Box>
      </CardContent>
     </Card >
    )}
export default Post
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ВЕБ-ЗАСТОСУНОК ОБМІНУ ПРОПОЗИЦІЯМИ ДЛЯ ДОПОМОГИ ВНУТРІШНЬО ПЕРЕМІЩЕНИМ ОСОБАМ

Програма та методика тестування

КПІ.ІТ-8216.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена СИРОТА

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Мар’яна ЛОЗЯК

Київ – 2022

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосунок обміну пропозиціями для допомоги ВПО, в якому користувачі можуть поширювати оголошення про можливі шляхи надання допомоги. Люди, що потребують підтримки навпаки можуть розміщувати повідомлення про потреби. Система представлена клієнтською частиною, що написана з використанням JavaScript-бібліотеки React, та серверною частиною, що створена з використанням сервісів, які надає Firebase.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення у відповідності до функціональних вимог;

– перевірка правильності роботи створених Firebase Function, що відповідають за роботу з БД і опрацьовують запити клієнта;

– перевірка роботи основних елементів інтерфейсу користувача;

– знаходження проблем та недоліків з метою їх усунення.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється уся документація, яка аналізується на предмет дотримання стандартів програмування;

– функціональне – перевіряється правильність виконання поставлених задач та відповідність усім вимогам до програмного продукту;

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується вручну, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення, так і в зручності в користуванні.

Для перевірки роботи застосунку необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– ручне тестування шляхом надсилання запитів до серверної частини через додаток Postman, які містять некоректні чи неповні дані;

– тестування на пристроях з різною роздільною здатністю екрану;

– тестування на пристроях з різною версією операційної системи;

– тестування роботи застосунку у різних вебпереглядачах;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування інтерфейсу користувача.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ВЕБ-ЗАСТОСУНОК ОБМІНУ ПРОПОЗИЦІЯМИ ДЛЯ ДОПОМОГИ ВНУТРІШНЬО ПЕРЕМІЩЕНИМ ОСОБАМ

Керівництво користувача

КПІ.ІТ-8216.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена СИРОТА

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Мар’яна ЛОЗЯК

Київ – 2022

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

«Potribni» – це вебзастосунок для розміщення оголошень про можливість надання допомоги чи про потребу. Основною метою створення застосунку було спрощення процесу об’єднання тих, кому потрібна підтримка і тих, хто може її надати.

Функціональними можливостями системи є:

- додавання нового оголошення;
- пошук по оголошеннях;
- фільтрація розміщених оголошень;
- видалення оголошення;
- можливість залишати коментарі;
- можливість додавати у «Вибране»;
- можливість редагувати профіль;
- можливість реєстрації та авторизації.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність комп’ютера або ноутбука з операційною системою Windows 7 або вище, macOS версії 10.0.0.3 або вище, мобільного пристрою з операційною системою на базі Android з версією 6.0 або вище, IOS версії 11 або вище;

– наявність доступу до Інтернету;

– наявність одного з веб-переглядачів Google Chrome, Mozilla Firefox, Safari, Opera, Microsoft Edge переважно останніх версій.

2.2 Завантаження застосунку

Для того, щоб працювати із застосунком потрібно в адресний рядок вебпереглядача ввести адресу застосунку або ж перейти за посиланням.

2.3 Перевірка коректної роботи

В разі, якщо після переходу за адресою веб застосунку користувач побачив головний екран або ж екран для входу, то робота із застосунком розпочата успішно.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

При вході користувачу буде відображено головний екран, на якому спершу відображаються пропозиції допомогти. Про це свідчить положення регулятора типів оголошення («Потребую»/»Хочу допомогти»)(рис. 3.1).

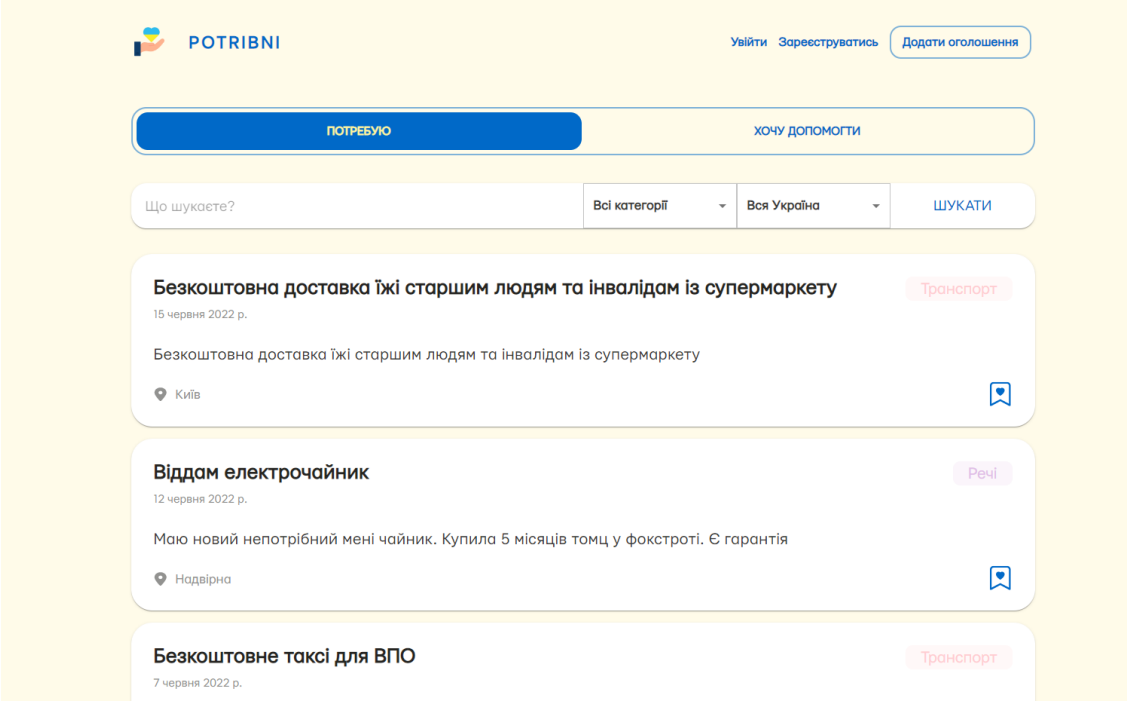


Рисунок 3.1 – Початкова сторінка додатку

Користувач має змогу переглянути доступні пропозиції. На кожній з карток-оголошень відображається назва, категорія, місце та стислий опис оголошення.

Для того, щоб побачити повідомлення про потреби, якщо ви бажаєте допомогти, слід встановити регулятор, що знаходиться одразу під навігаційною панеллю, у відповідне положення (написи зроблені зі сторони користувача: я хочу допомогти).(рис.3.2)

Для того, щоб повернутись на головний екран з будь-якої точки застосунку можна натиснути на логотип на верхній панелі навігації.

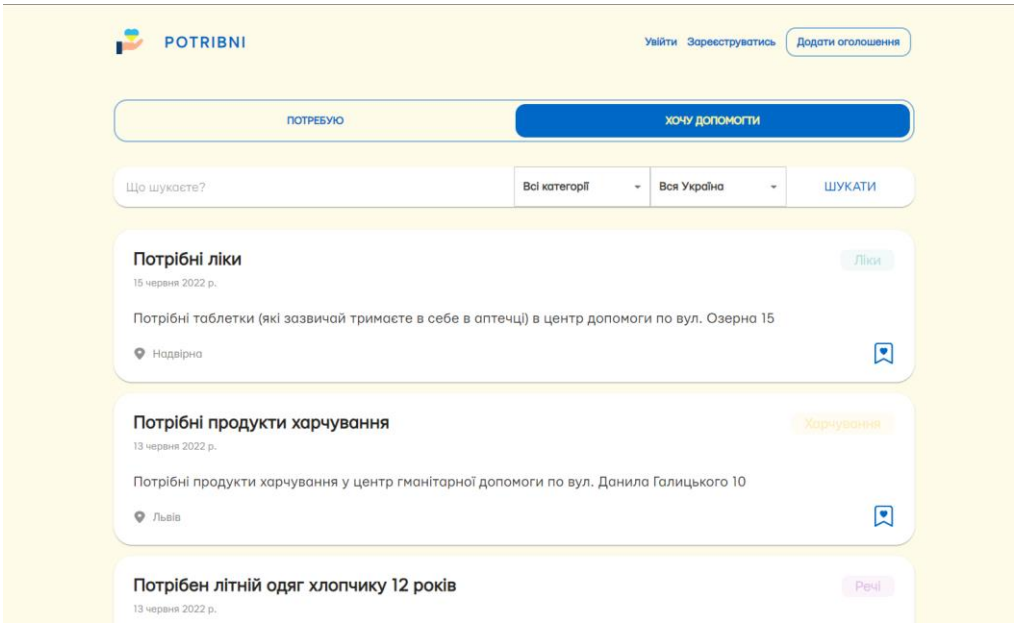


Рисунок 3.2 – Перегляд оголошень про потребу

Для того, щоб детальніше переглянути оголошення достатньо натиснути на карту із ним. В результаті буде доступний перегляд повного опису оголошення, інформації про користувача, що його розмістив, та коментарі до обраної публікації.

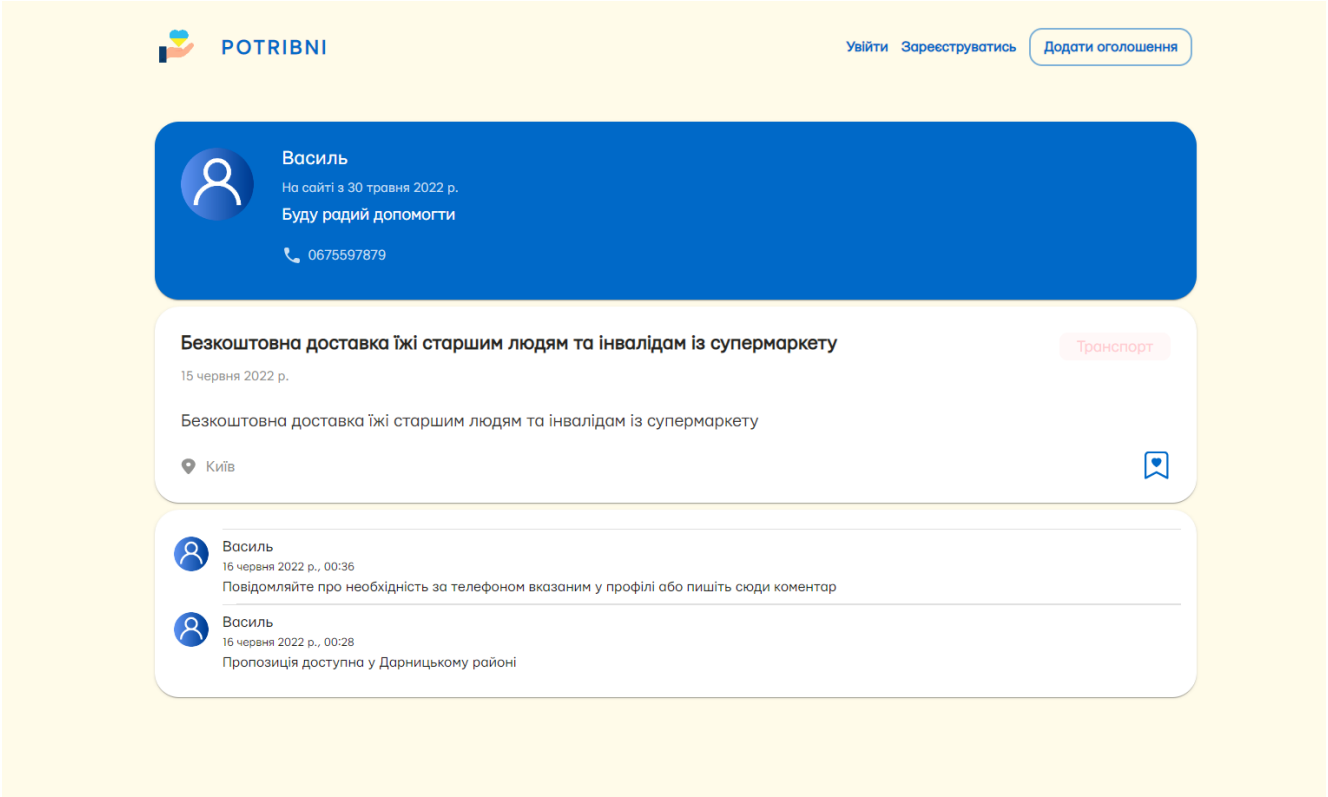


Рисунок 3.3 – Перегляд вікна з окремим оголошенням

На даному етапі можливість коментування оголошень чи додавання їх у «Вибране» недоступні.

Для того, щоб переглянути інформацію про користувача та список всіх його оголошень необхідно натиснути на синю картку з інформацією користувача на попередньому вікні (рис.3.4).

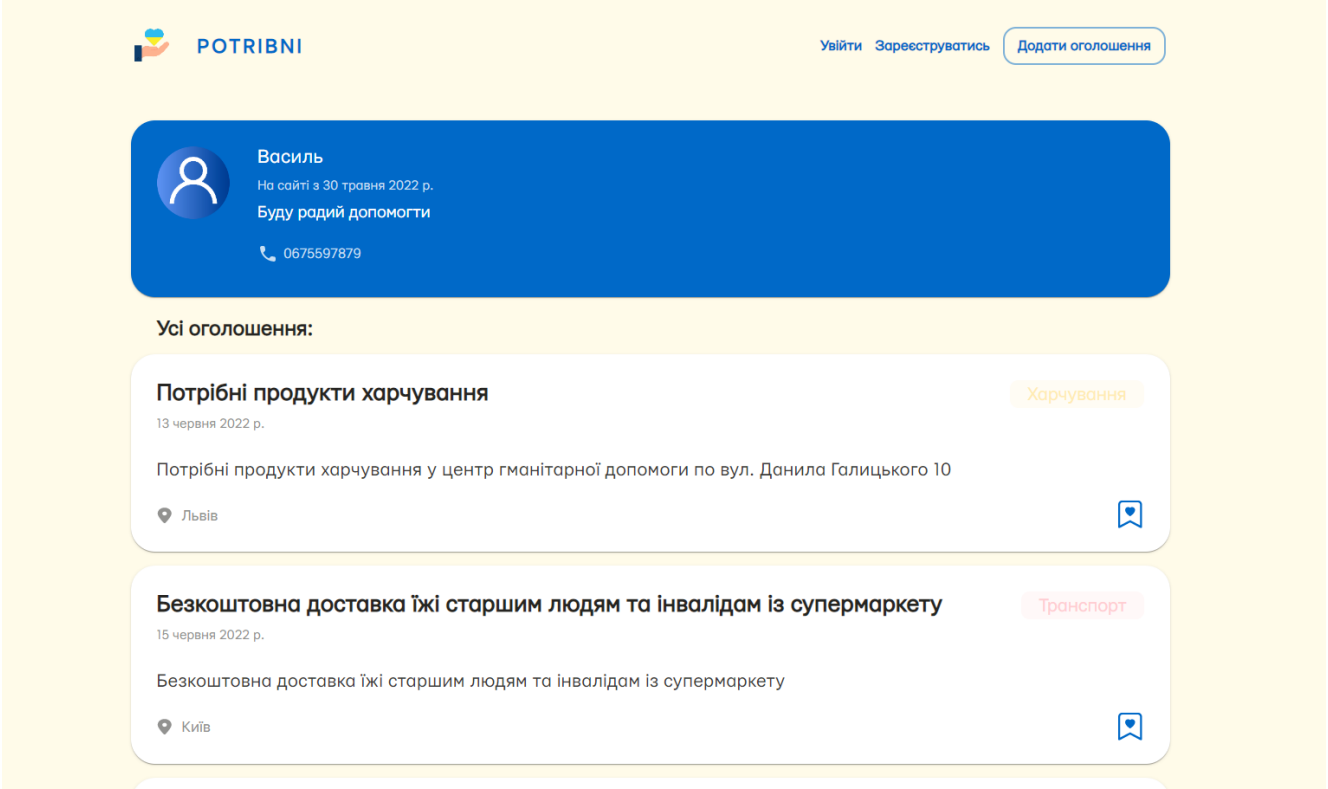


Рисунок 3.4 – Перегляд інформації про користувача

Для того, щоб здійснити пошук серед оголошень певного типу (потреба/пропозиція) необхідно на головному вікні у полі пошуку вказати текст, пошук на вміст якого у назвах та описах буде проведено, та натиснути кнопку «Шукати». При цьому можна не обирати місце та категорію. Тоді пошук буде здійснено лише за текстом (рис.3.5).

Можна також здійснити пошук враховуючи вміст тексту та відповідність певним категоріям (рис. 3.6). Залишивши одне з полів для вибору у стані за замовчуванням користувач автоматично задає можливість відповідності всім можливим значенням місць чи категорій (рис. 3.7).

Для того, щоб відфільтрувати оголошення достатньо залишати поле з текстом пошуку пустим (рис. 3.8).

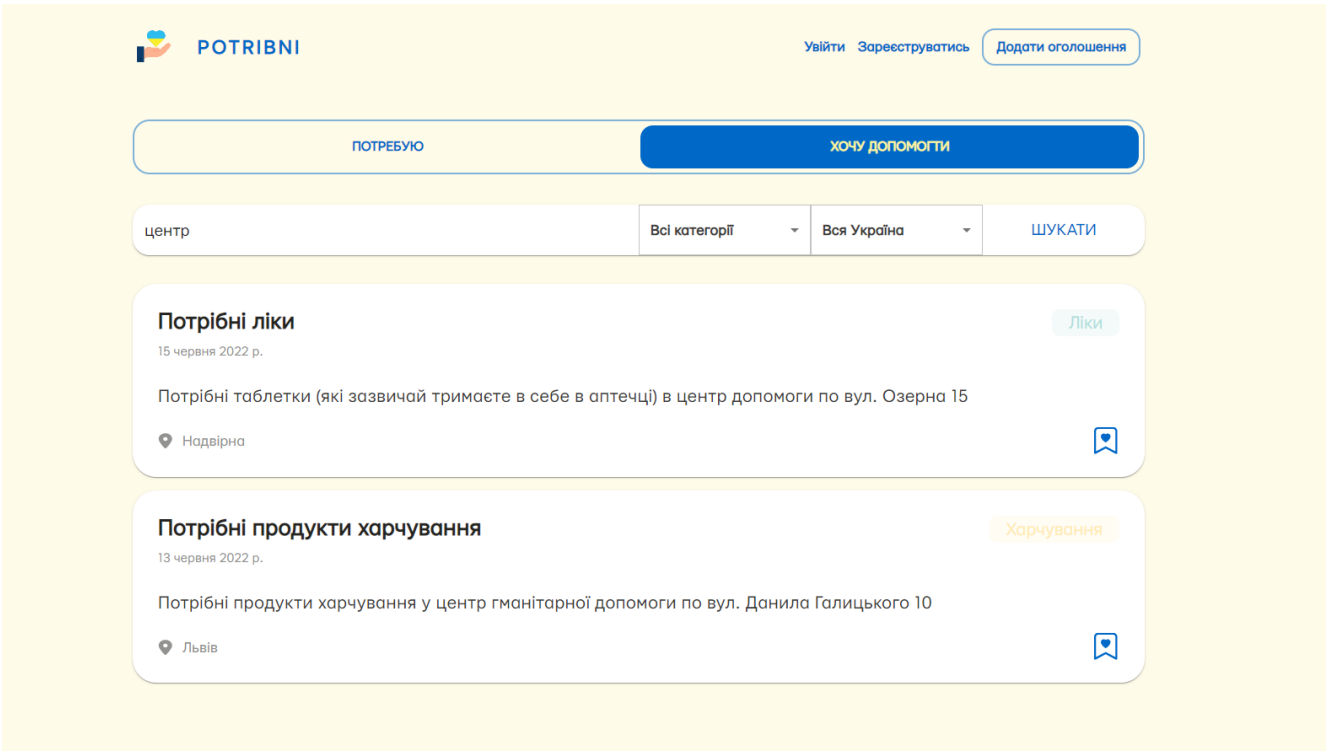


Рисунок 3.5 – Пошук лише за вмістом тексту у назві чи описі оголошення

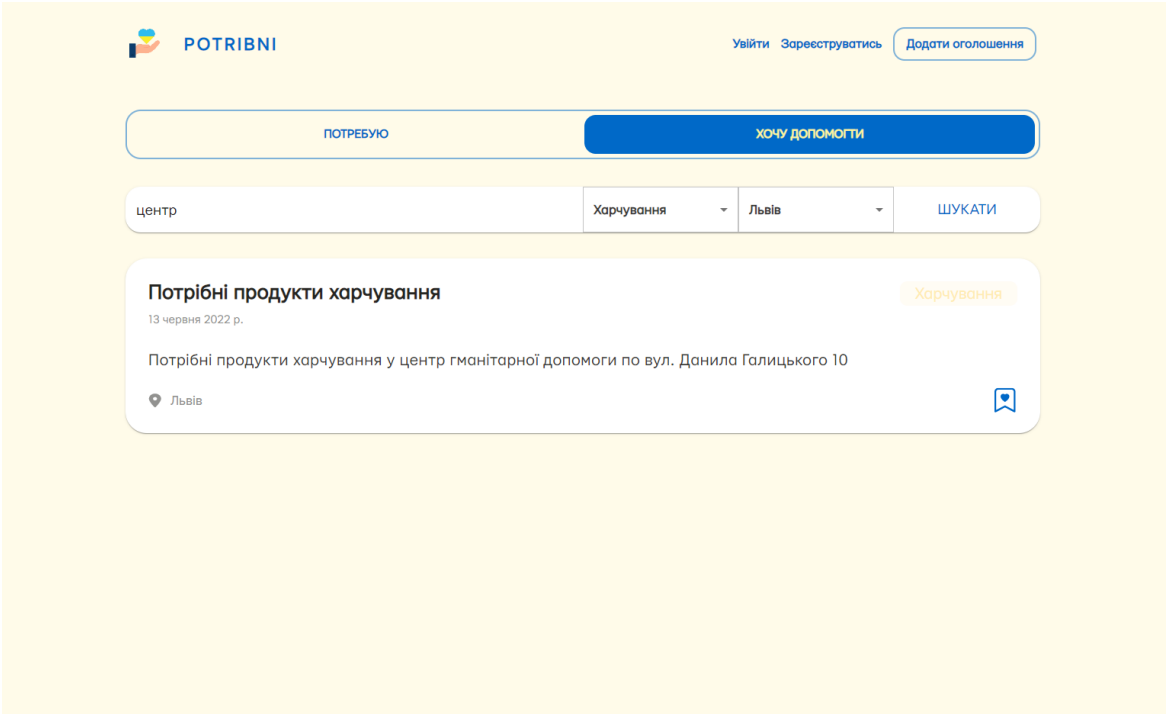


Рисунок 3.6 – Пошук за вмістом тексту та відповідністю місцю та категорії

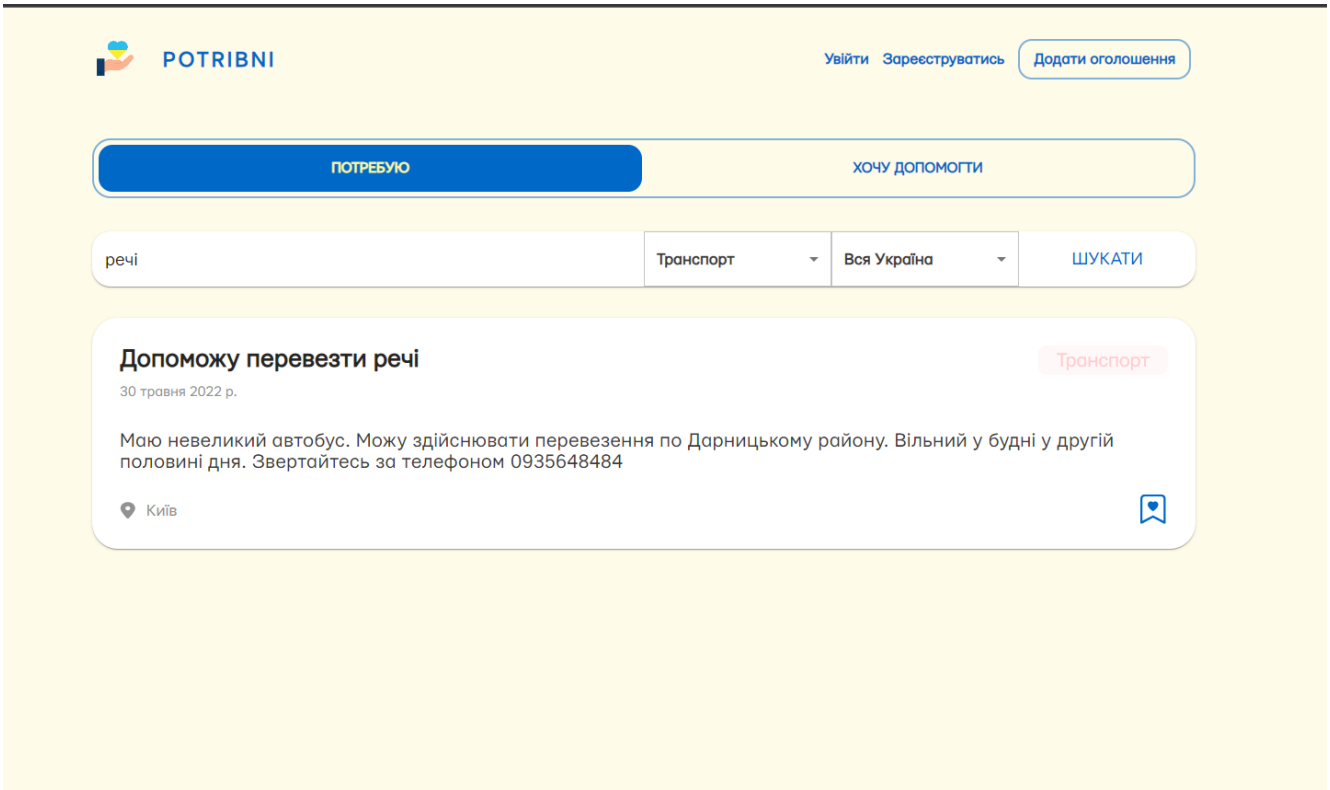


Рисунок 3.7 – Пошук за текстом та категорією

Для здійснення фільтрації поле для введення тексту пошукового запиту залишено пустим:

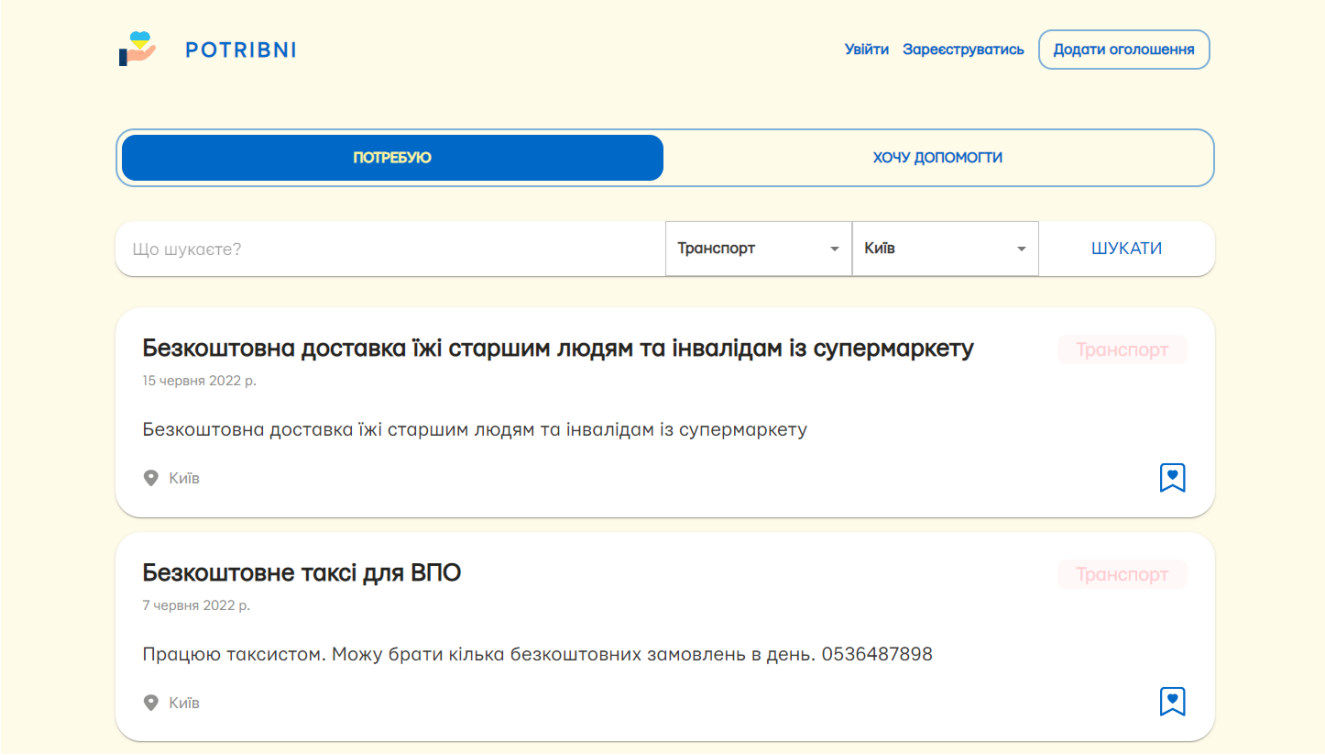


Рисунок 3.8 – Фільтрація оголошень

Для того, щоб мати можливість користуватись більшою кількістю функцій, які надає система, необхідно увійти або зареєструватись. Для цього можна натиснути на одну з відповідних кнопок, розташованих у правому верхньому куті екрану на навігаційній панелі екрану. На кожному з вікон, що з'являються є поля для вводу необхідної інформації, а також посилання на сторінку входу (якщо під час реєстрації користувач згадав, що вже має акаунт) і навпаки посилання на сторінку реєстрації із сторінки входу (рис.3.9).

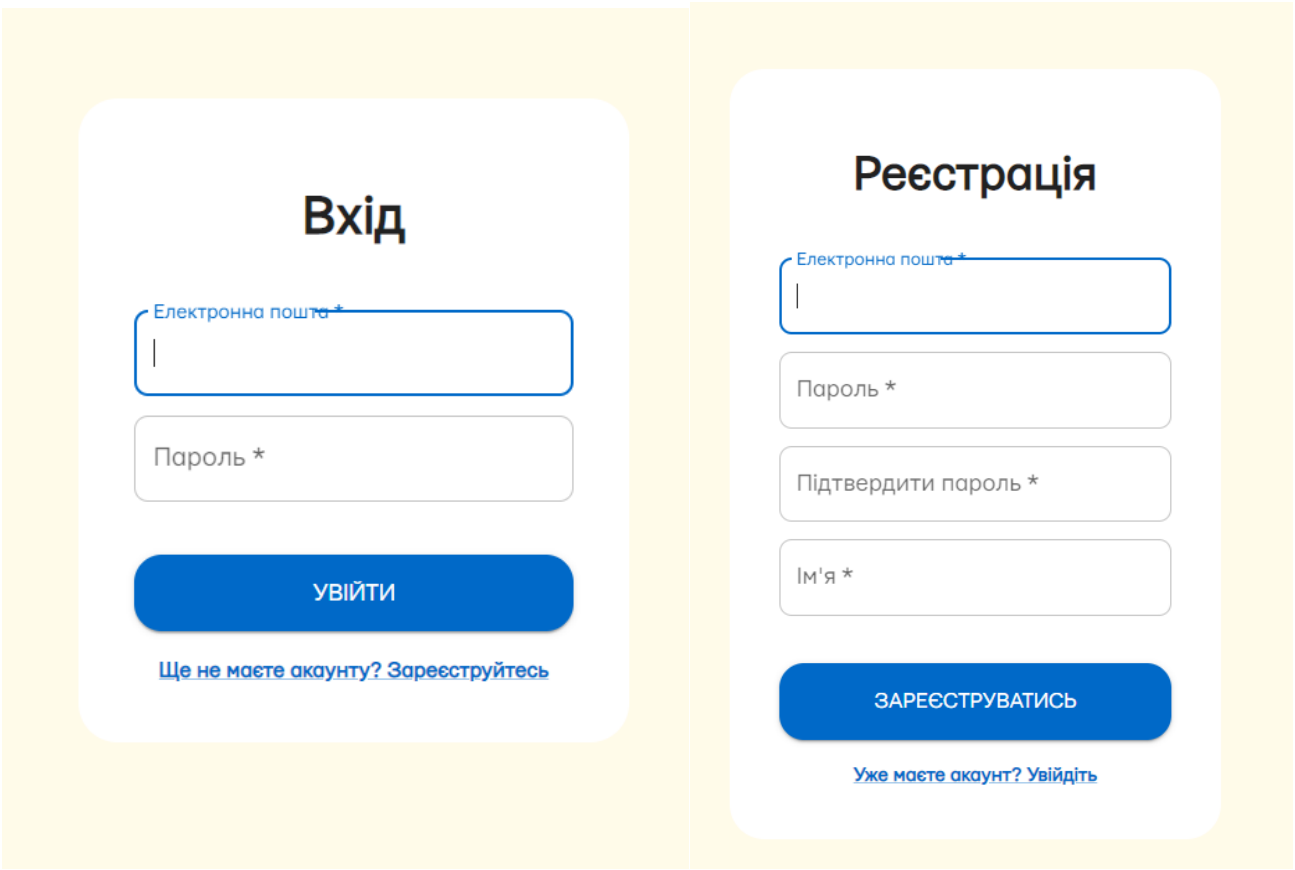


Рисунок 3.9 – Екрани входу та реєстрації

У випадках, якщо дані було введено невірно, з'являтимуться графічні підказки з пояснювальним текстом (рис.3.10).

Такими помилками можуть бути, наприклад, залишені пусті поля чи неправильні дані для входу.

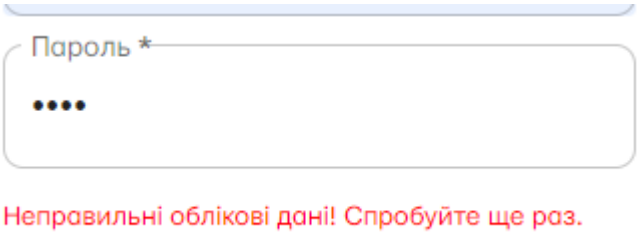


Рисунок 3.10 – Приклад відображення помилки

Після входу в систему навігаційна панель змінює вигляд. На ній розміщуються ім’я користувача, кнопка для перегляду вибраних оголошень, кнопка для виходу з системи та кнопка «Додати оголошення».

Щоб залишити коментар до будь-якого оголошення достатньо натиснути на карту-оголошення, в полі введення написати коментар та натиснути кнопку «Коментувати». Коментар з’явиться першим у списку.

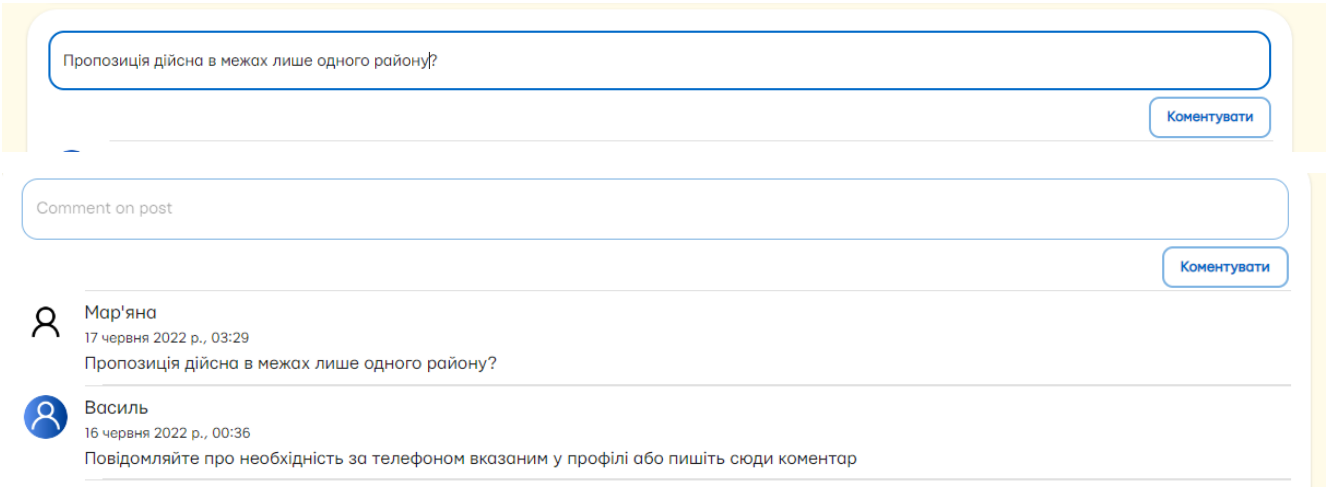


Рисунок 3.11 – Процес додавання коментаря

Чи не найголовнішою функцією системи є можливість додавати нове оголошення. Для цього необхідно на верхній панелі натиснути кнопку «Додати оголошення» після чого, буде відкрита форма для внесення даних. Серед них: відмітка про тип інформації(пропозиція/потреба), вибір категорії, місця, поля для введення назви та опису оголошення. Після внесення всіх необхідних даних слід натиснути на кнопку «Додати» (або «Відмінити»)(рис.3.12). Після створення

оголошення відображається екран з детальною інформацією опублікованого повідомлення.

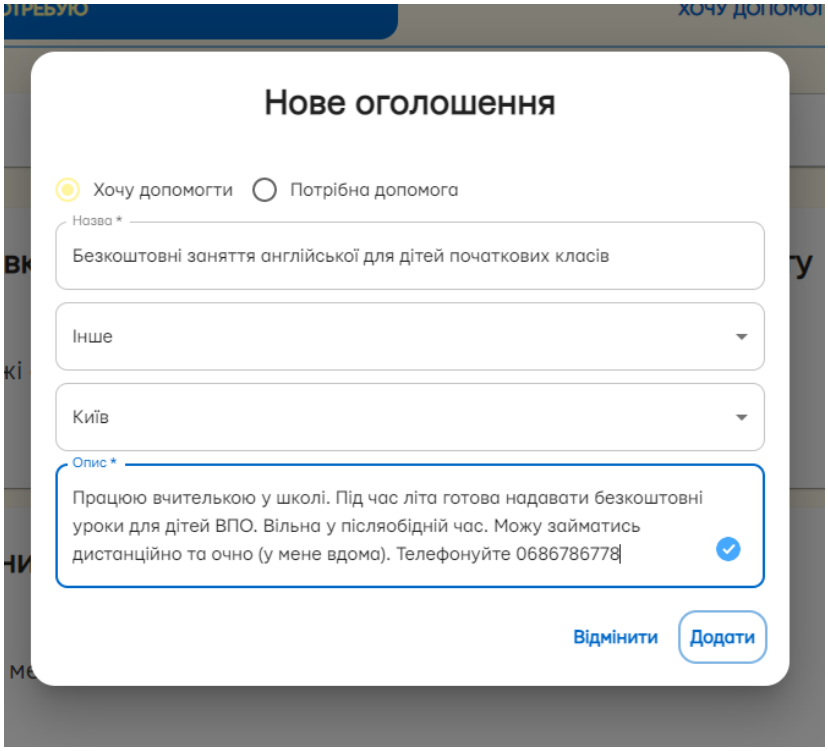


Рисунок 3.12 – Додавання нового оголошення

Для того, щоб видалити повідомлення необхідно натиснути на відповідну іконку у правому нижньому куті картки оголошення, лише за умови, що воно було опубліковано поточним користувачем. Після натискання з'являється діалогове вікно з питанням чи користувач впевнений, що бажає видалити оголошення, і двома кнопками: «Відмінити» і «Підтвердити».

Рисунок 3.13 – Карта оголошення з кнопкою видалення

Якщо оголошення було опубліковано не поточним користувачем, то замість кнопки видалення відображається кнопка із зображенням закладки, що дозволяє

додати його до колекції «Вибране». Кнопка відображається зафарбованою, якщо оголошення вже було обрано.

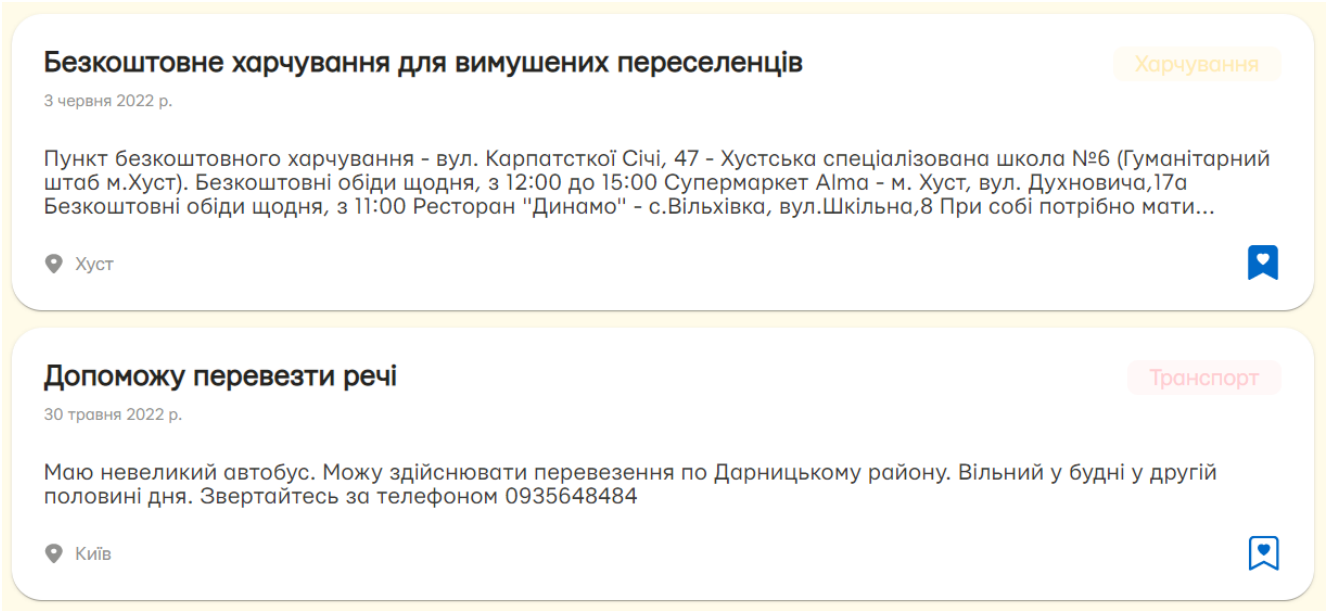


Рисунок 3.14 – Оголошення додане та недодане у «Вибране»

Для того, щоб переглянути всі обрані оголошення необхідно натиснути на кнопку із зображенням книжкової закладки на навігаційній панелі. В результаті буде відображено список обраних оголошень (назва та дата, коли повідомлення було додано до колекції) (рис.3.15).

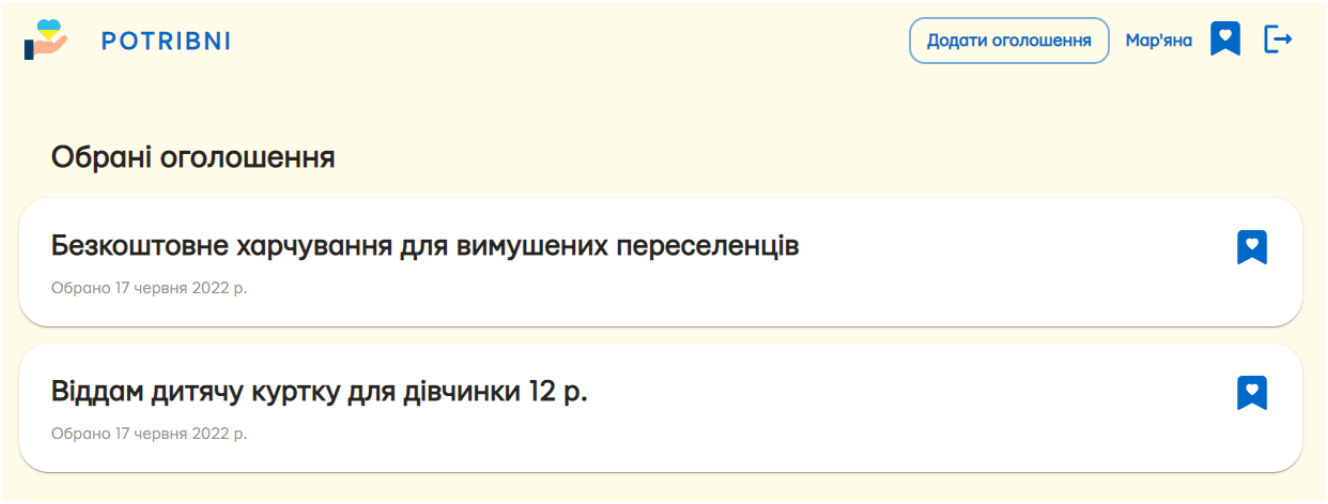


Рисунок 3.15 – Вигляд екрану для перегляду обраних оголошень

Для того, щоб переглянути інформацію про себе, свій список оголошень чи відредагувати (додати) інформацію про себе, потрібно натиснути на ім’я

користувача у верхній правій частині панелі навігації. В результаті буде відображено сторінку поточного користувача (рис.3.16).

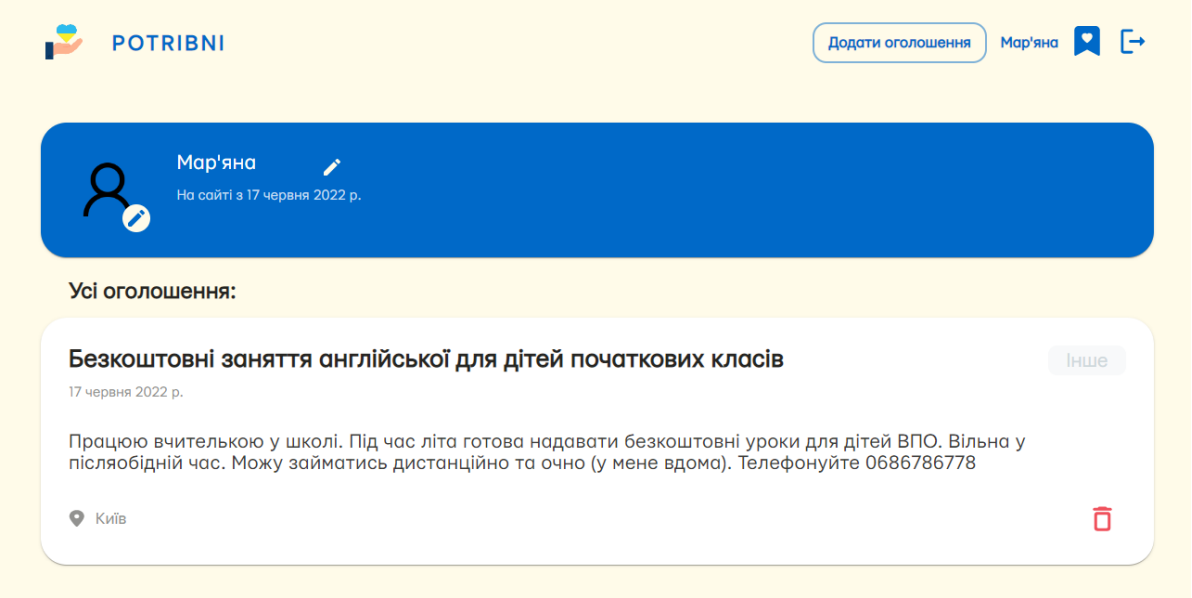


Рисунок 3.16 – Екран для перегляду власного профілю

Для того, щоб змінити фото достатньо натиснути на олівець біля зображення користувача. Відкриється вікно вибору файлу. Далі слідує доволі зрозуміла процедура вибору зображення. Для того, щоб додати чи змінити додаткову інформацію про себе, користувач може натиснути на кнопку-олівець біля основного вмісту профілю. Відкриється вікно редагування з полями для внесення опису, міста перебування та контактного номеру телефону. А також з кнопками для підтвердження чи скасування внесення змін.

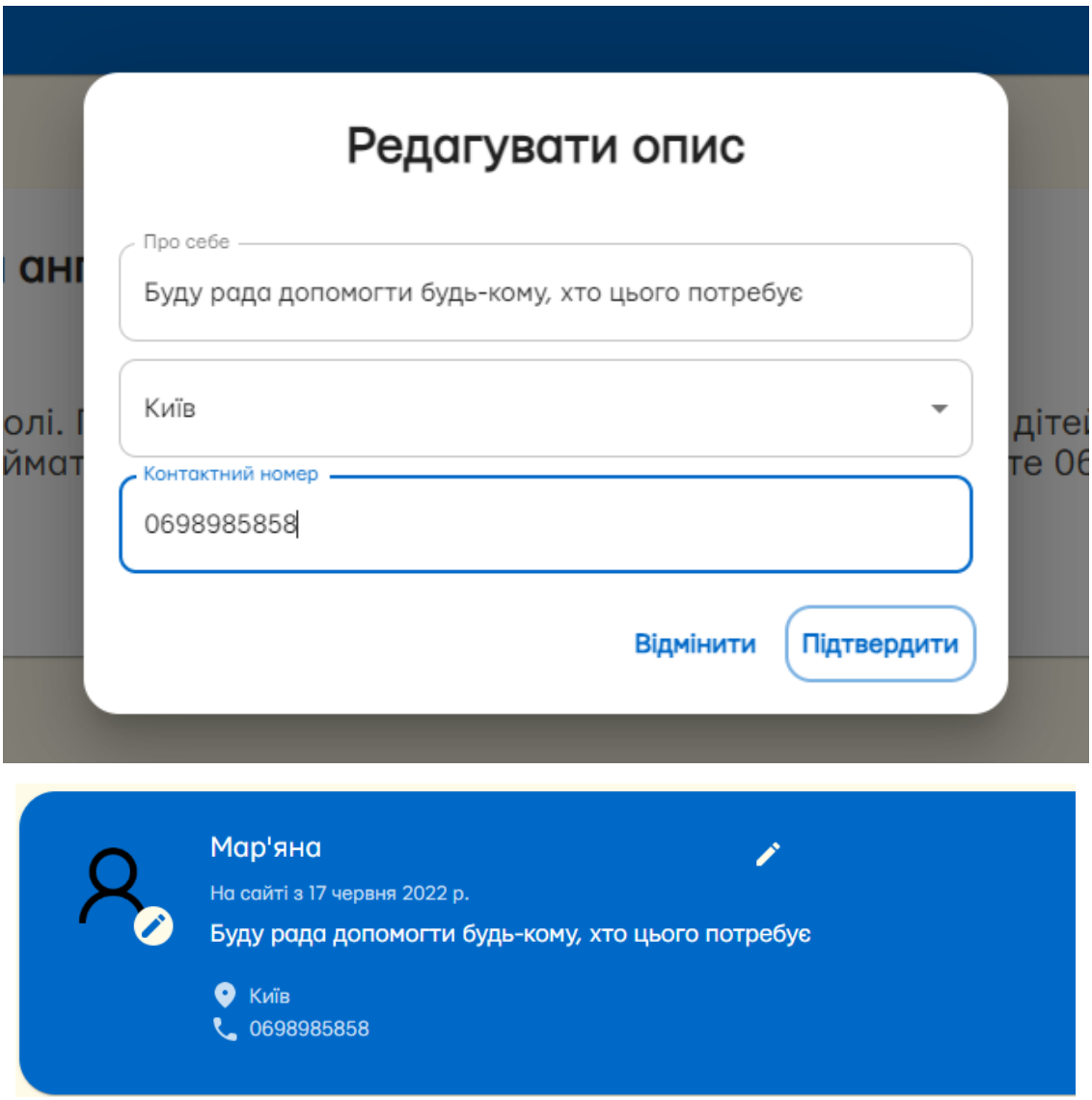


Рисунок 3.17 – Форма та результат внесення змін до опису профілю

Для того, щоб вийти з даного акаунту достатньо натиснути на кнопку «Вихід», що знаходиться у правому верхньому куті екрану.

Для того, щоб завершити роботу із вебзастосунком достатньо просто покинути сторінку.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ВЕБ-ЗАСТОСУНОК ОБМІНУ ПРОПОЗИЦІЯМИ ДЛЯ ДОПОМОГИ ВНУТРІШНЬО ПЕРЕМІЩЕНИМ ОСОБАМ

Графічний матеріал

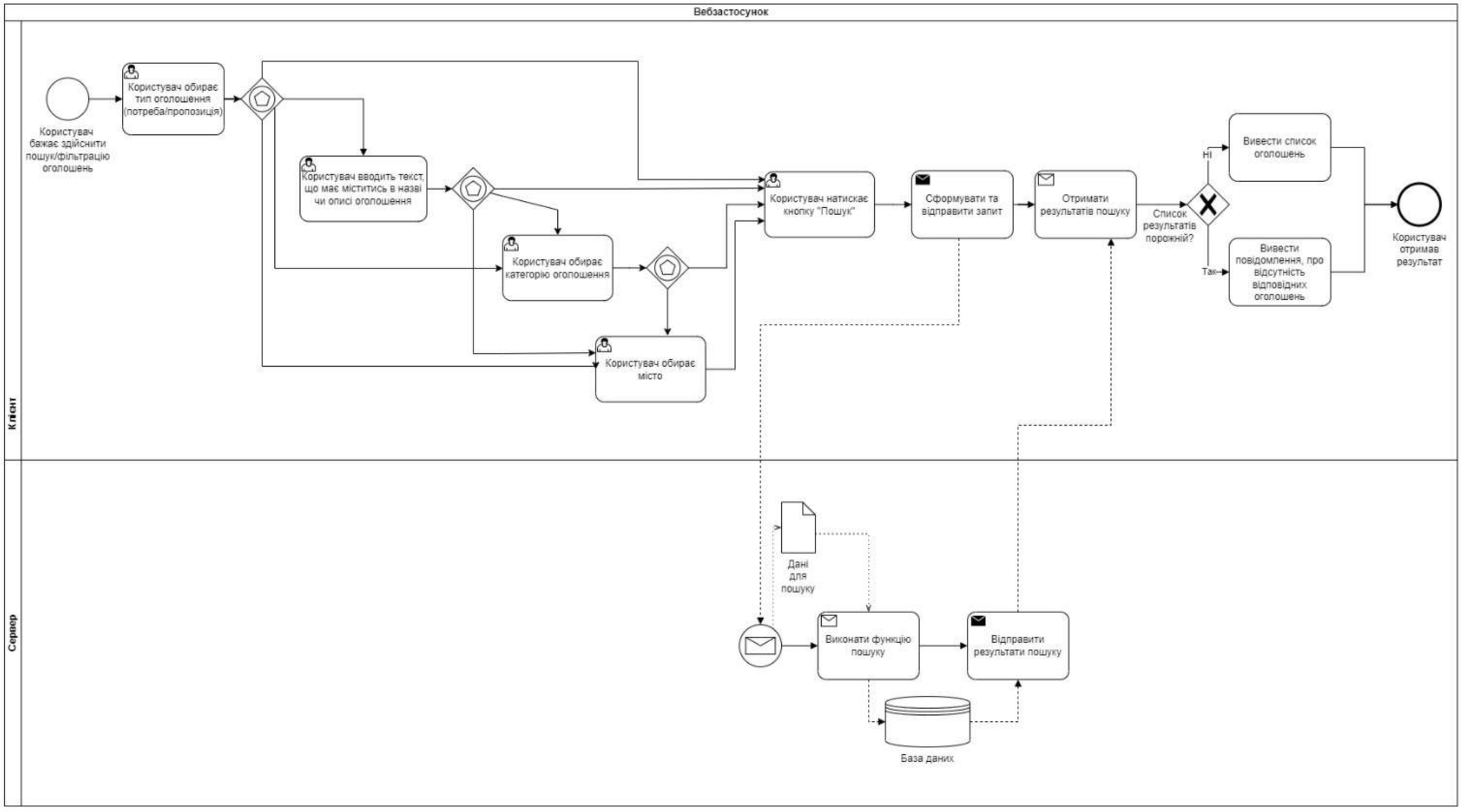
КПІ.ІТ-8216.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена СИРОТА

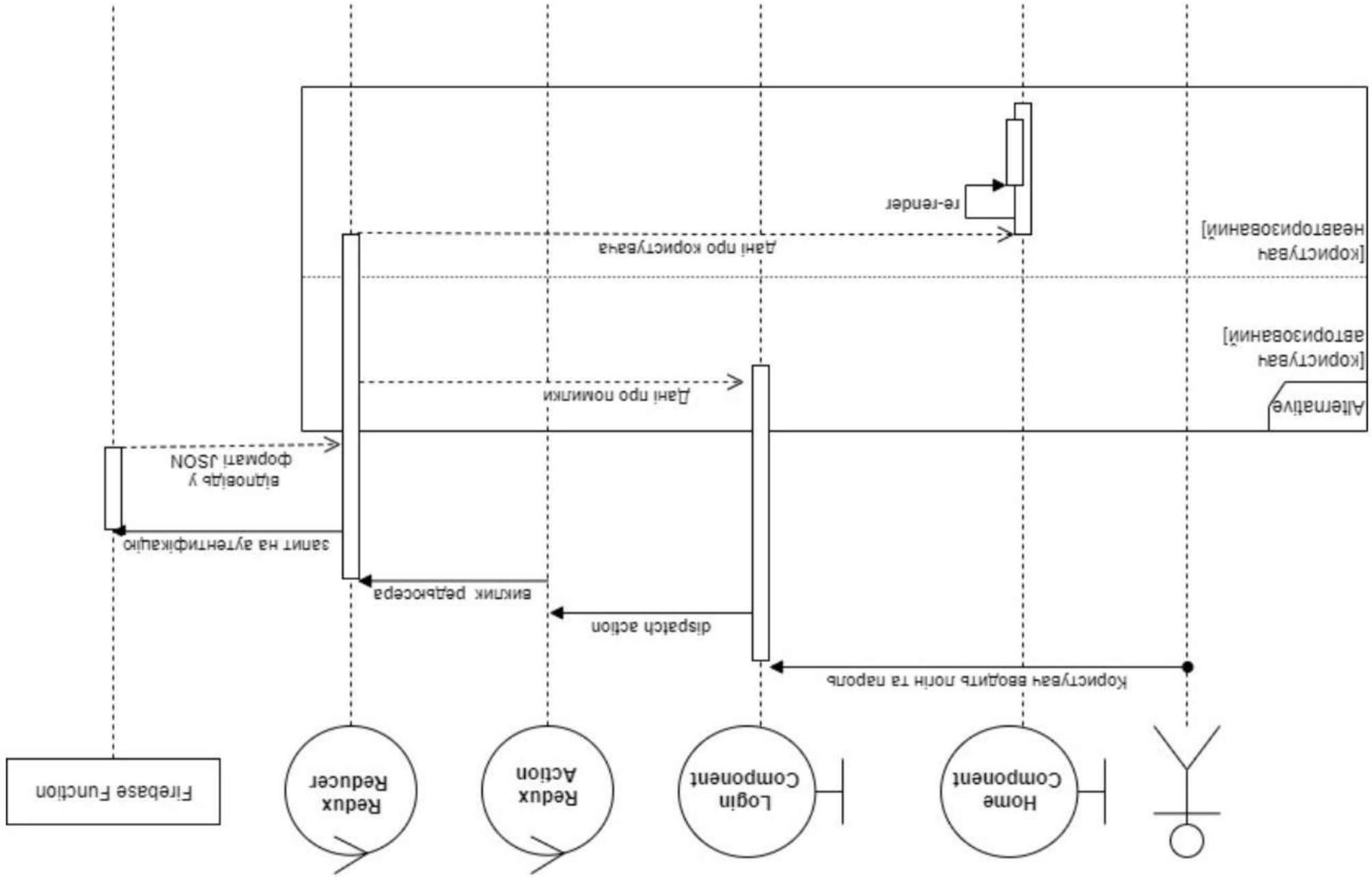
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Мар’яна ЛОЗЯК

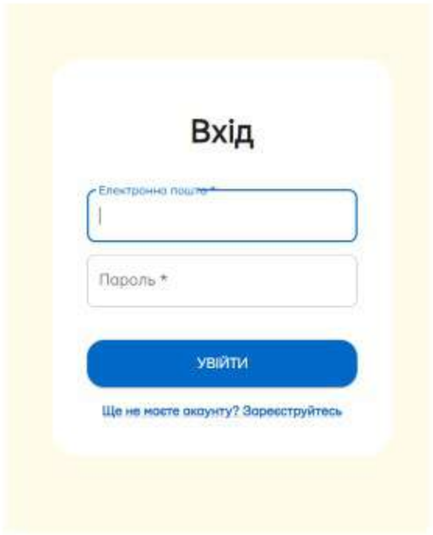
Київ – 2022

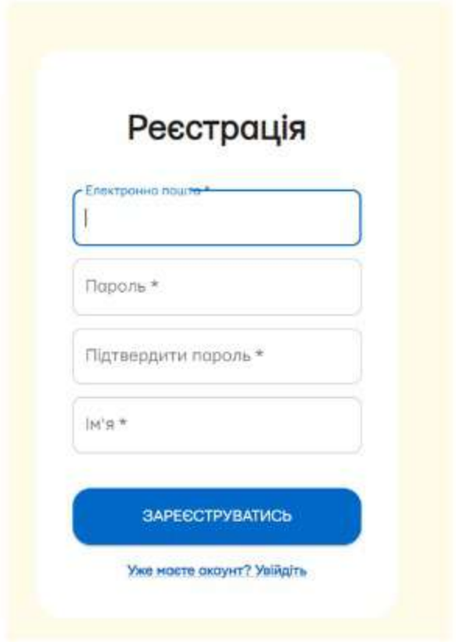


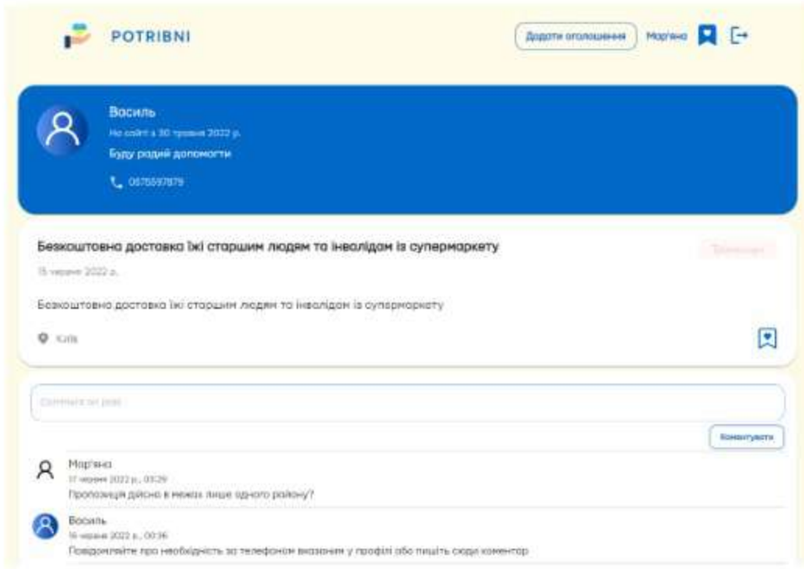
					КПІ.ІТ-8216.045440.06.99.СБП	
					Схема бізнес-процесу	Літера: Маса: Масштаб:
Зм.	Арк.	№ документа	Підпис	Дата		Аркуш: Аркушів:
Розробив		Лозяк М.В.				
Перевірив		Сирота О.П.				
Т. кон.						
					Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-82
Н. кон.		Ліщук К.І.				
Затвердив		Сирота О.П.				

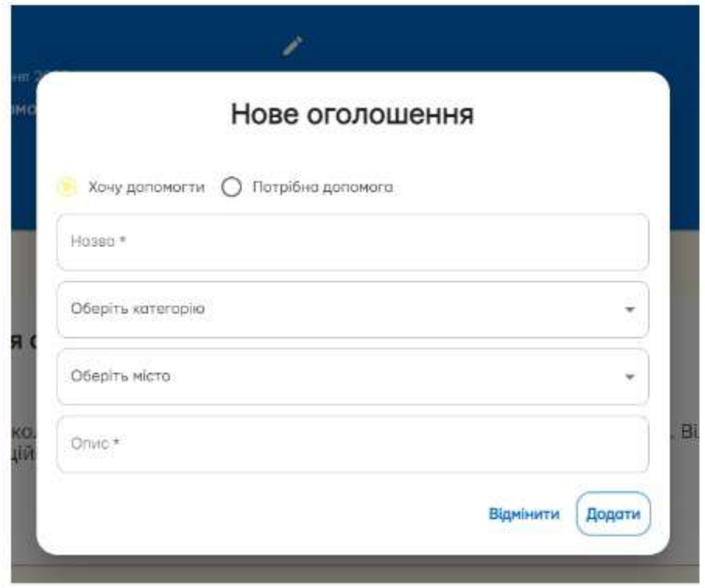


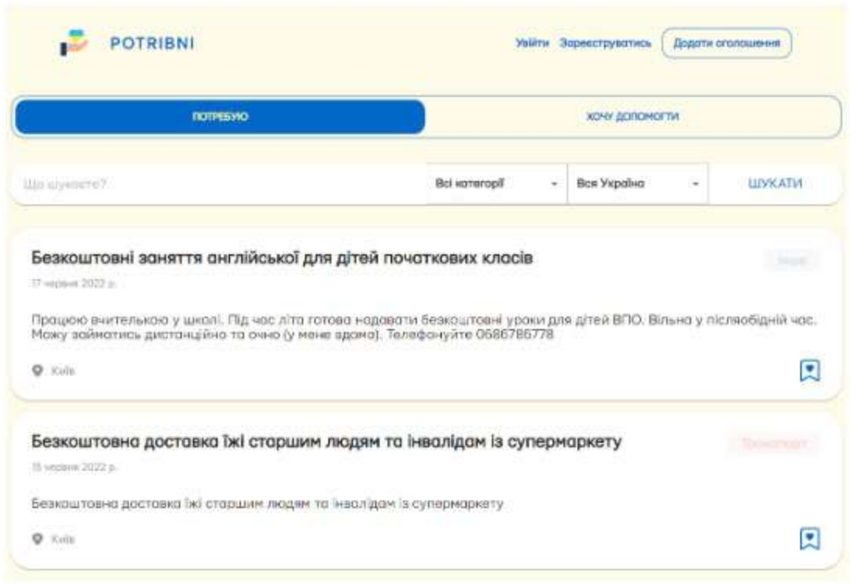
Зм.	Арк.	№ документа	Підпис	Дата	КПІ.ІТ-8216.045440.06.99.ССП								
Розробив		Лозяк М.В.											
Перевірив		Сирота О.П.											
Т. кон.													
Н. кон.		Пшук К.І.											
Затвердив		Сирота О.П.											
Літера		Маса		Масштаб									
Арк.ув				Аркушів									
КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІТ-82					Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам								
					Структурна схема послідовностей								

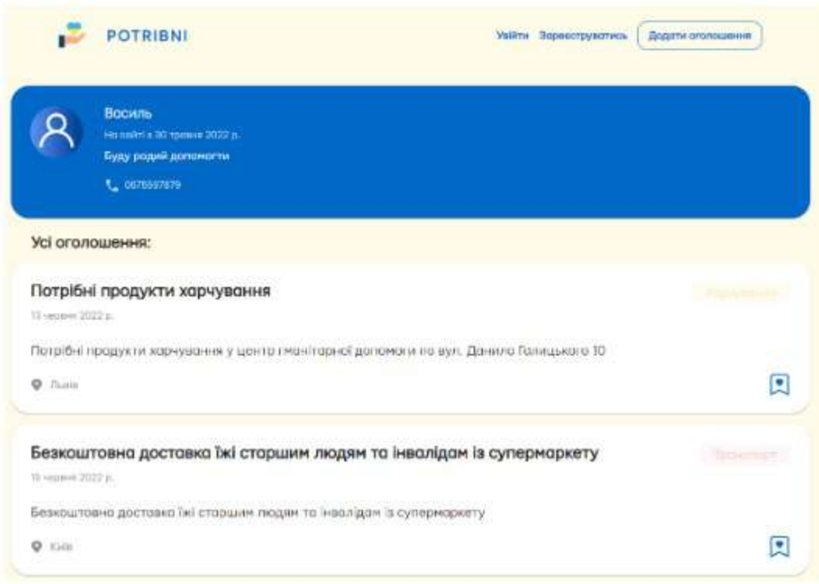


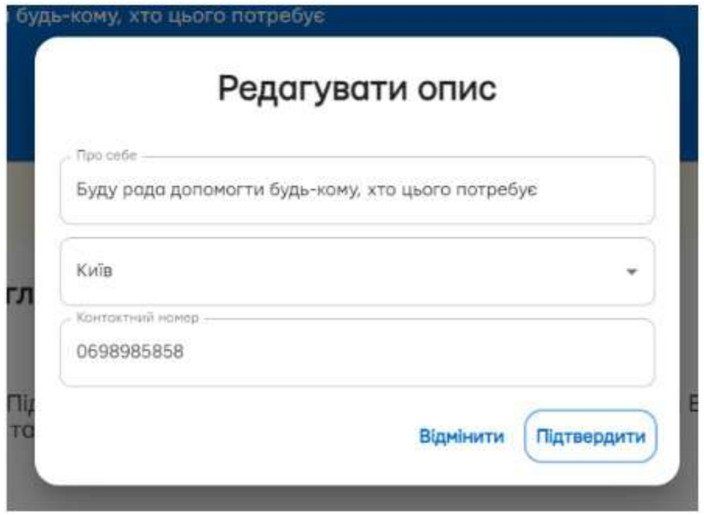


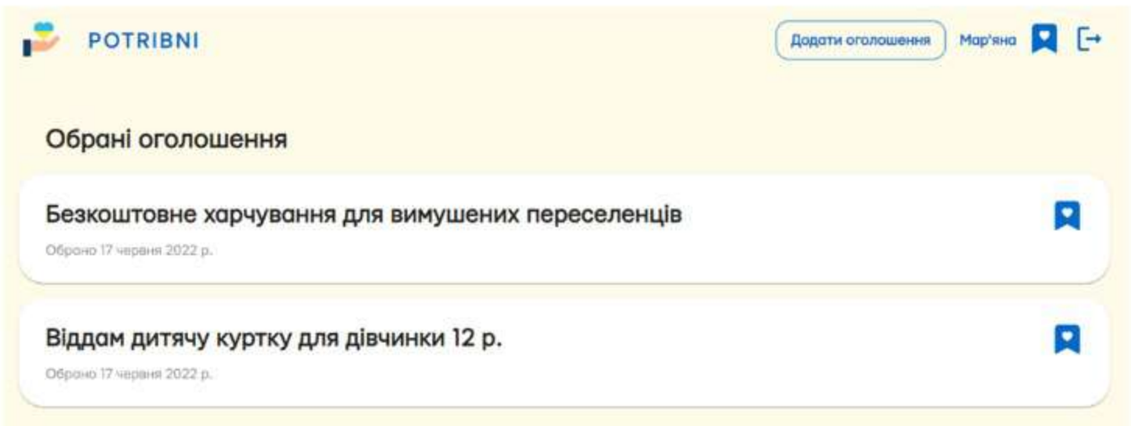












					КПІ.ІТ-8216.045440.06.99.КЕ			
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм			
Розробив		Лозяк М.В.				Аркуш	Аркушів	
Перевірив		Сирота О.П.						
Т. кон.								
					Веб-застосунок обміну пропозиціями для допомоги внутрішньо переміщеним особам	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-82		
Н. кон.		Ліщук К.І.						
Затвердив		Сирота О.П.						

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1011475348

Дата проверки:
06.06.2022 17:40:12 EEST

Тип проверки:
Doc vs Internet + Library

Дата отчета:
06.06.2022 17:41:53 EEST

ID пользователя:
76913

Название файла: **IT-82_Лозяк_ПЗ**

Количество страниц: **59** Количество слов: **9665** Количество символов: **74479** Размер файла: **1.11 MB** ID файла: **1011353170**

2.07%
Совпадения

Наибольшее совпадение: **0.43%** с источником из Библиотеки (ID файла: **1008233072**)

0.92% Источники из Интернета	18	Страница 61
1.97% Источники из Библиотеки	117	Страница 61

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников