МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

**КУРСОВА РОБОТА**

з курсу: «Об’єктно-орієнтоване програмування»

на тему: «Розроблення програмного забезпечення з використанням

об’єктно-орієнтованої парадигми»

Виконав: студент (ка) 2 курсу групи 621п

Спеціальність 121 – Інженерія програмного забезпечення

(код та найменування)

Гордеюк Р. О.

(прізвище й ініціали студента(ки))

Керівник: Лучшев П.О.

(прізвище й ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_ Лучшев П.О.

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_ Дем’яненко В.А.

(підпис) (прізвище та ініціали)

Харків – 2024

**ТИПОВЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

Варіант 7. Логістичні послуги

|  |  |  |
| --- | --- | --- |
| **Критерій оцінювання** | **Бали** | **Штрафи** |
| **Покрокове виконання курсової роботи:** | | |
| Розділ 1(крайній термін – **3 тиждень**) | 0..15 | -3 |
| Розділ 2(крайній термін – **6 тиждень**) | 0..15 | -3 |
| Розділ 3(крайній термін – **9 тиждень**) | 0..15 | -3 |
| Розділ 4(крайній термін – **12 тиждень**) | 0..15 | -3 |
| Розділ 5(крайній термін – **13 тиждень**) | 0..5 | -1 |
| Оформлення пояснювальної записки (термін – **14 тиждень**) | 0..3 | -1 |
| **Підготовка до захисту курсової роботи:** | | |
| доповідь з презентацією (термін – **15 тиждень**) | 0..1 |  |
| відеоролик з демонстрацією роботи програми  (термін – **15 тиждень**) | 0..1 |  |
| **Захист курсової роботи:** | | |
| доповідь з презентацією | 0..5 |  |
| демонстрація роботи програми і github-репозиторію | 0..5 |  |
| відповіді на питання | 0..5 |  |
| використання\*:   1. інкапсуляції 2. статичних членів класів 3. інтерфейсів 4. абстрактних класів 5. спадкування 6. поліморфізму 7. .NET-delegates і events 8. сереалізації/десереалізації об’єктів 9. Collections.Generic 10. LINQ   \* мають бути представленні на захисті курсової роботи у презентації | 0..15 |  |
| **Всього за курсову роботу:** | **0..100** | **0..14** |

**Зміст**

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 2](#_Toc165849902)

[ВСТУП 2](#_Toc165849903)

[1 ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc165849904)

[1.1 Аналіз функціональності програмних аналогів 2](#_Toc165849905)

[1.2 Побудова Mind Map для заданої предметної області 2](#_Toc165849906)

[1.3 Виділення ролей користувачів і формулювання функціональних вимог 2](#_Toc165849907)

[1.4 Проєктування інтерфейсу користувача 2](#_Toc165849908)

[1.5 Розроблення функціональних тестів 2](#_Toc165849909)

[1.5.1 Функціональні тести для ролі користувача «Адміністратор» 2](#_Toc165849910)

[1.5.2 Функціональні тести для ролі користувача «Зареєстрований користувач» 2](#_Toc165849911)

[1.5.3 Функціональні тести для ролі користувача «Гість» 2](#_Toc165849912)

[2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ 2](#_Toc165849913)

[2.1 Виділення і опис класів предметної області 2](#_Toc165849914)

[2.2 Встановлення зв’язків між класами 2](#_Toc165849915)

[3 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ 2](#_Toc165849916)

[3.1 Структура проєкту з реалізацією класів предметної області 2](#_Toc165849917)

[3.2 Реалізація інтерфейсів і каркасів класів предметної області 2](#_Toc165849918)

[3.3 Розроблення unit-тестів для класів предметної області 2](#_Toc165849919)

[3.4 Повна реалізація класів предметної області 2](#_Toc165849920)

[3.5 Результати unit-тестування класів предметної області 2](#_Toc165849921)

[4 ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА 2](#_Toc165849922)

[4.1 Структура проєкту з реалізацією класів інтерфейсу користувача 2](#_Toc165849923)

[4.2 Виділення класів для реалізації інтерфейсу користувача 2](#_Toc165849924)

[4.3 Програмна реалізація класів інтерфейсу користувача 2](#_Toc165849925)

[5 ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 2](#_Toc165849926)

[5.1 Функціональне тестування для ролі користувача «Адміністратор» 2](#_Toc165849927)

[5.2 Функціональне тестування для ролі користувача «Зареєстрований користувач» 2](#_Toc165849928)

[5.3 Функціональне тестування для ролі користувача «Гість» 2](#_Toc165849929)

[ВИСНОВКИ 2](#_Toc165849930)

[ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ 2](#_Toc165849931)

[ДОДАТОК А. Лістинг класів предметної області 2](#_Toc165849932)

[ДОДАТОК Б. Лістинг класів і інтерфейсів 2](#_Toc165849933)

[ДОДАТОК В. Лістинг класів unit-тестів 2](#_Toc165849934)

[ДОДАТОК Г. Лістинг класів інтерфейсу користувача 2](#_Toc165849935)

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

|  |  |  |
| --- | --- | --- |
| CRUD | – | (*англ*. Create Read Update Delete) основні функції управління даними «створення, читання, оновлення і вилучення». |
| ASP.NET Core MVC | – | фреймворк для побудови веб-додатків на платформі .NET Core |
| Entity Framework | – | ORM, який надає доступ до бази даних у формі об'єктів, дозволяючи взаємодіяти з базою даних через об'єктно-орієнтований підхід |
| ООП | – | об’єктно-орієнтоване програмування |
| БД | – | база даних |
| ЗК | – | Зареєстрований користувач |

# ВСТУП

Із розвитком інтернету, тема онлайн-шопінгу стає більш популярною, що призводить до зростання кількості посилок, які потрібно доставити до клієнтів. Зі зростанням тенденції до збільшення соціальної дистанції, люди все частіше віддають перевагу замовленню товарів онлайн, ніж до особистого візиту в магазин.

Серед потенційних користувачів логістичних послуг можна виділити: бізнеси (малий, середній та великий), організації (урядові установи, благодійні та неприбуткові організації), інтернет-платформи, онлайн-покупці а також звичайні користувачі (використання послуг доставки для відправлення листів, подарунків чи інших особистих речей). Українськими лідерами серед послуг доставки є «Нова Пошта», «Укрпошта», «Meest», «Dhl», «Delivery» та інші, якщо брати онлайн-магазин який доставляє посилки через свої точки, то це «Rozetka».

Під час роботи над курсовою роботою буде використано такий стек-технологій: ASP.NET Core MVC, Entity Framework, БД – я буду використовувати SQL Server для зберігання даних.

# ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Аналіз функціональності програмних аналогів

1. Нова пошта

Сервіс дозволяє створювати та відслідковувати посилки. Присутня можливість реєстрації та авторизації на сайті. Також повністю розписано можливості роботи із сервісом для більш поглибленої взаємодії користувача [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)].

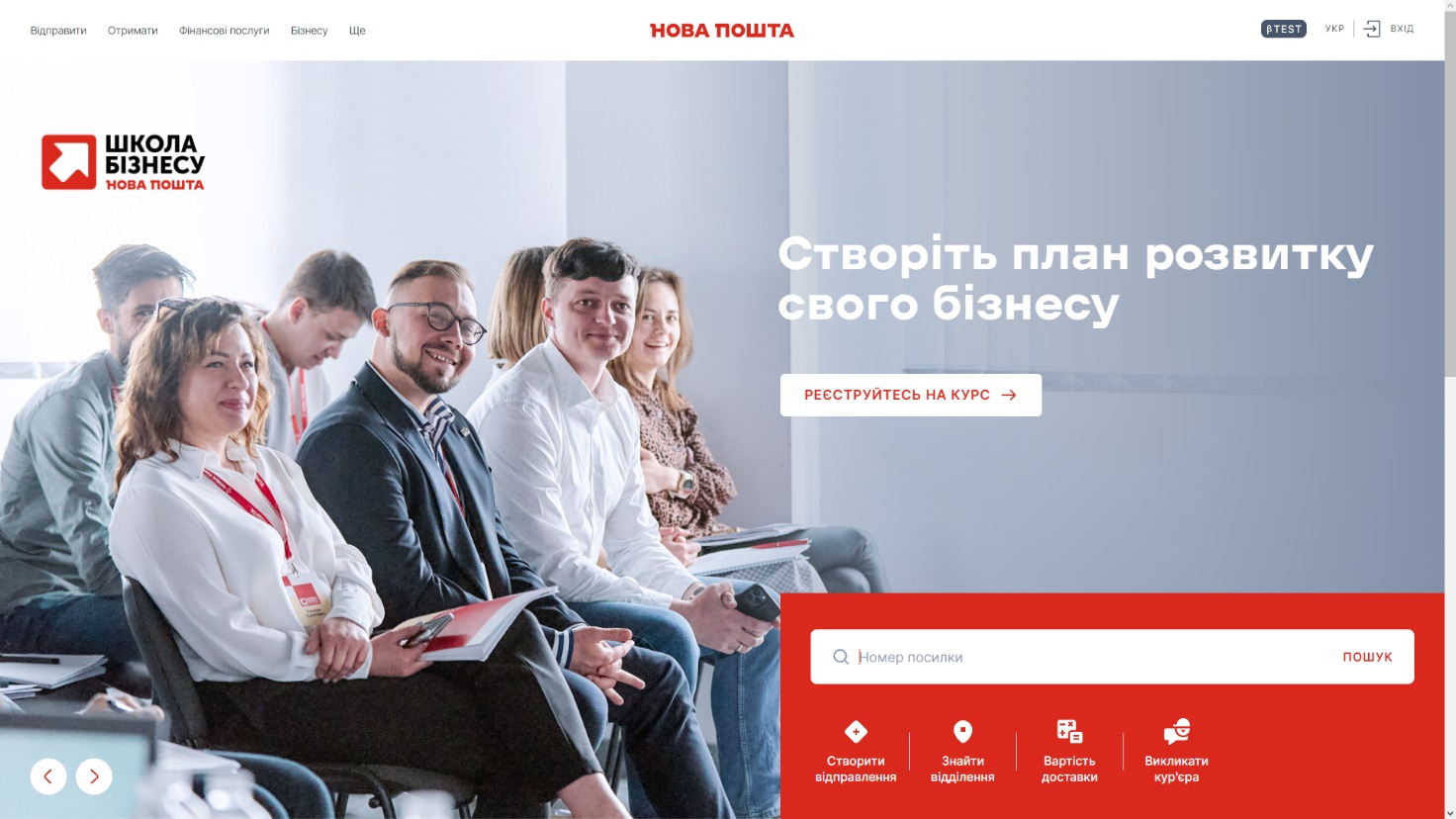


Рисунок 1.1 – Головна сторінка сайту «Нова Пошта»

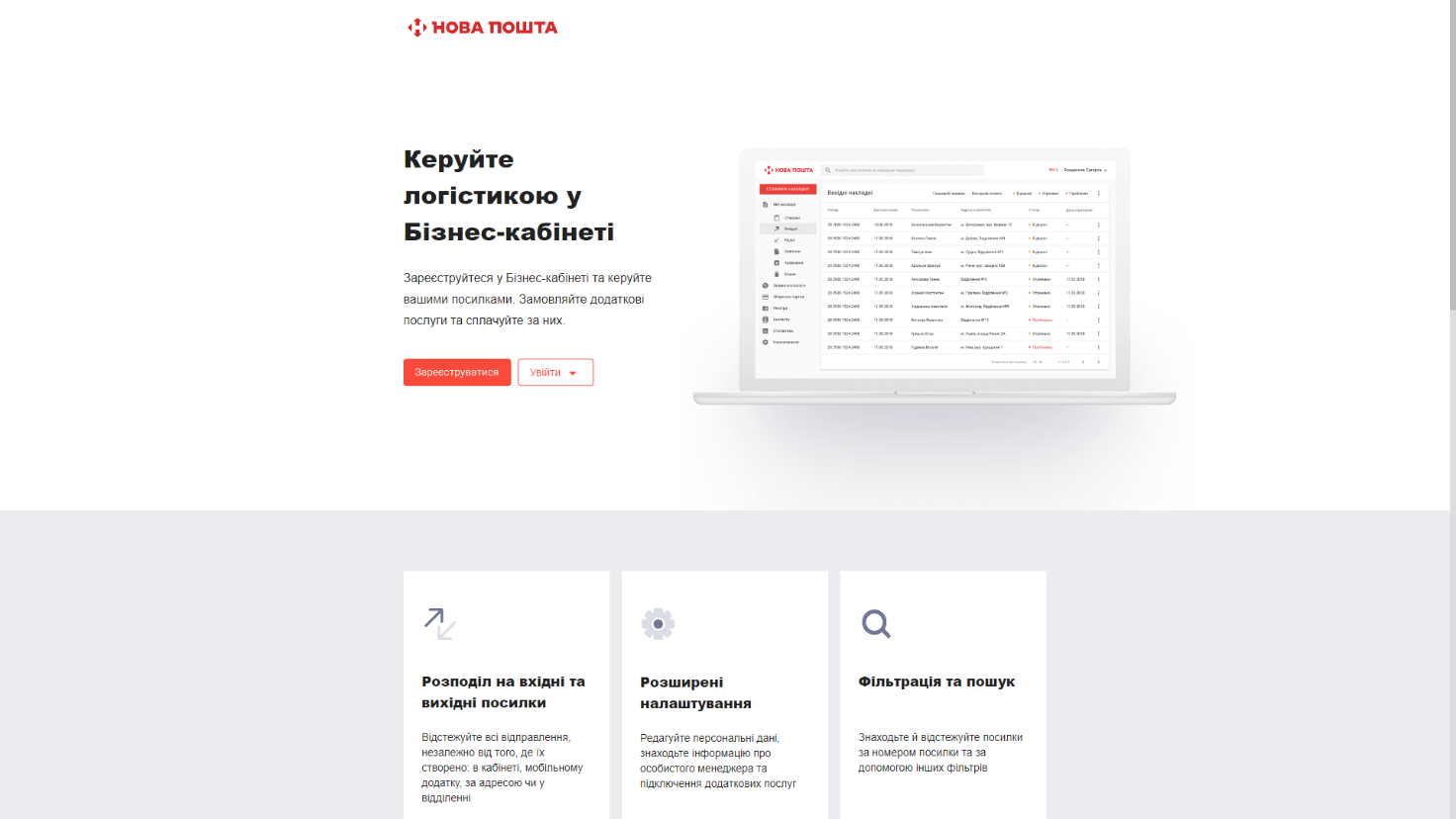


Рисунок 1.2 – Сторінка з реєстрацією/ авторизацією

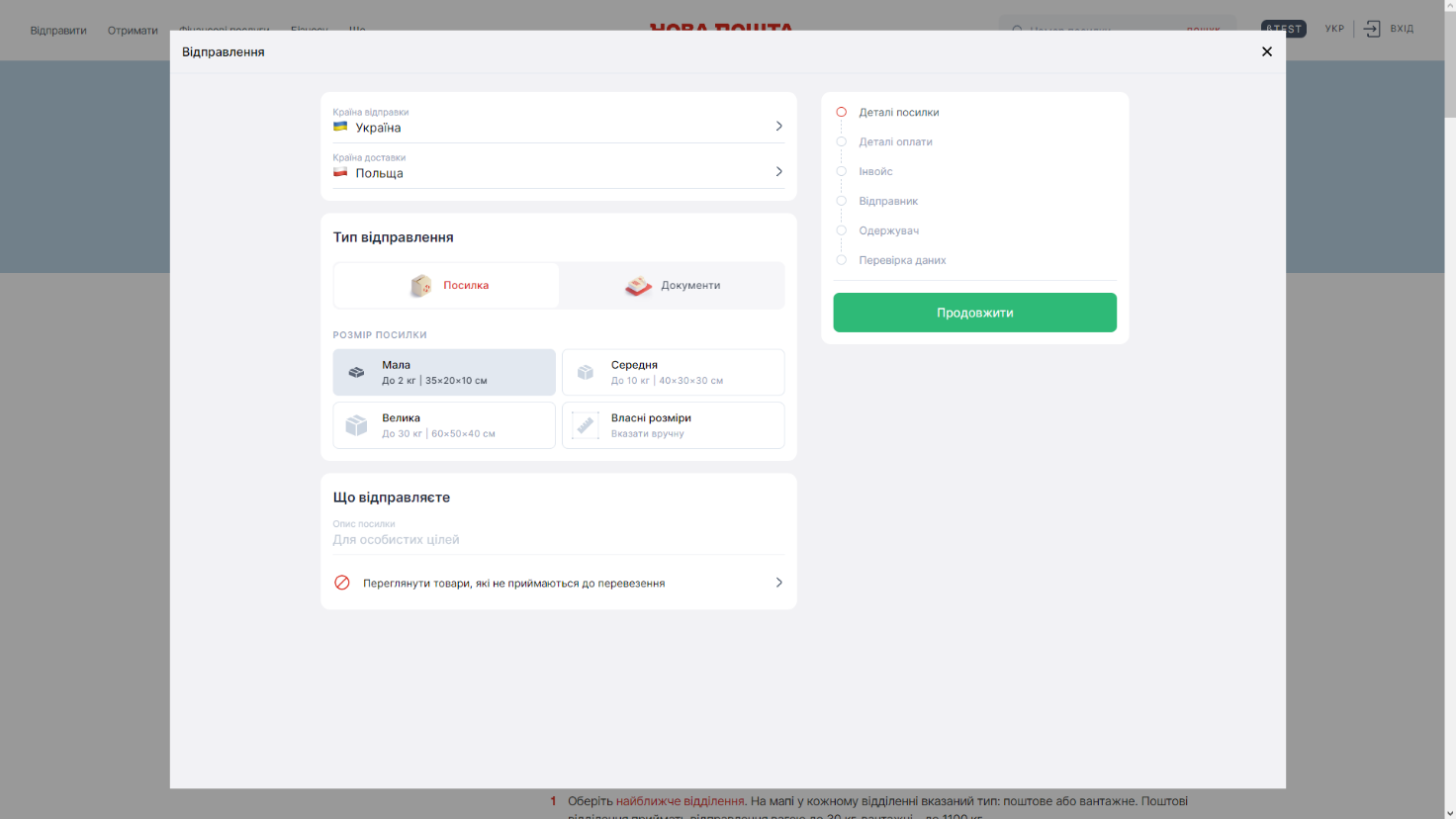


Рисунок 1.3 – Створення посилки

1. Укрпошта

Сервіс дозволяє зайти в особистий профіль користувача попередньо зареєструвавшись/ авторизувавшись. На сайті є можливість покупки онлайн товарів, відслідковування та оформлення посилок, розрахунок вартості доставки, пошук поштових індексів. Також є фінансові послуги, такі як грошові перекази або страхування [[2](#_ПЕРЕЛІК_ДЖЕРЕЛ_ТА)].

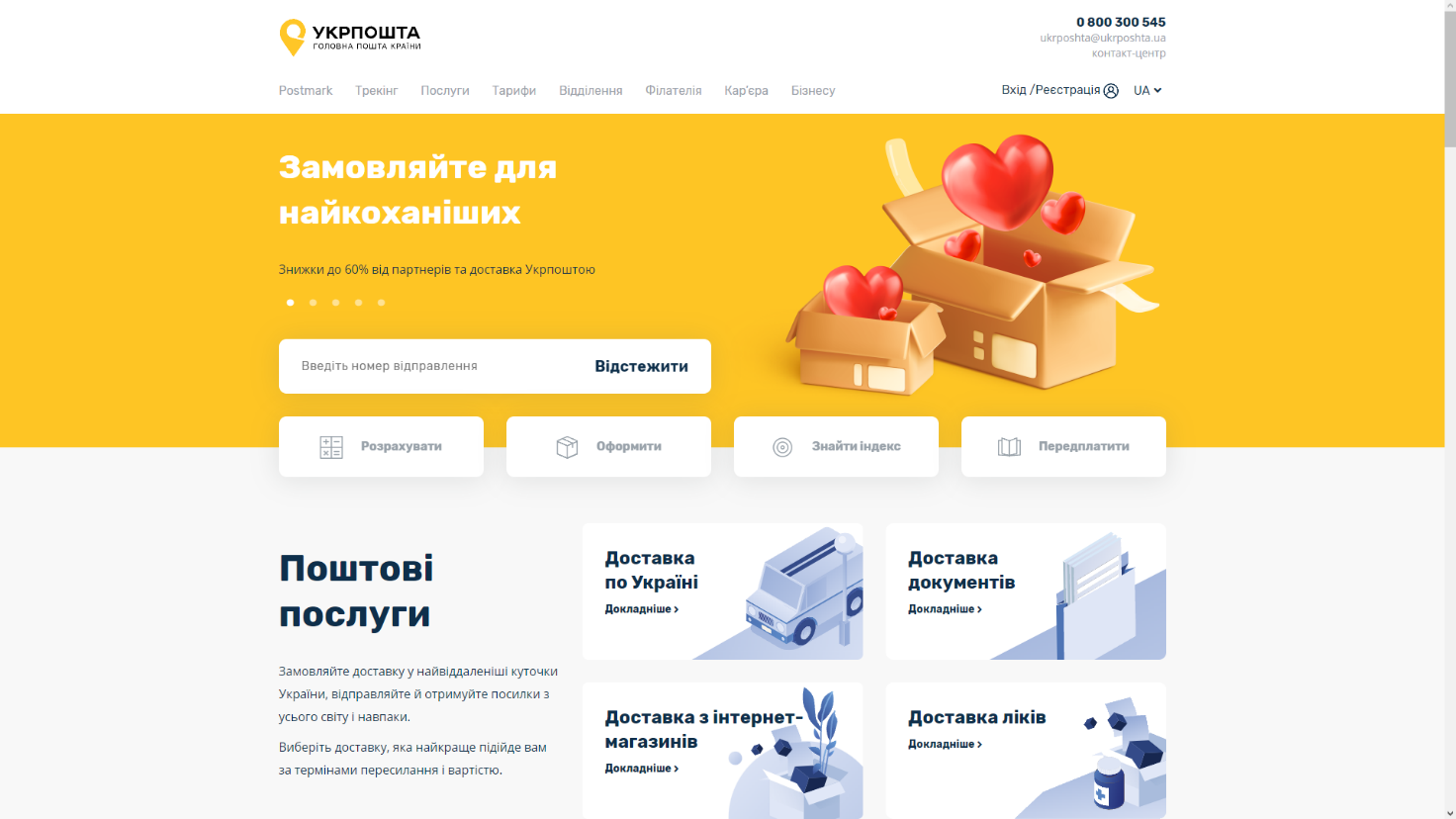


Рисунок 1.4 – Головна сторінка сайту «Укрпошта»

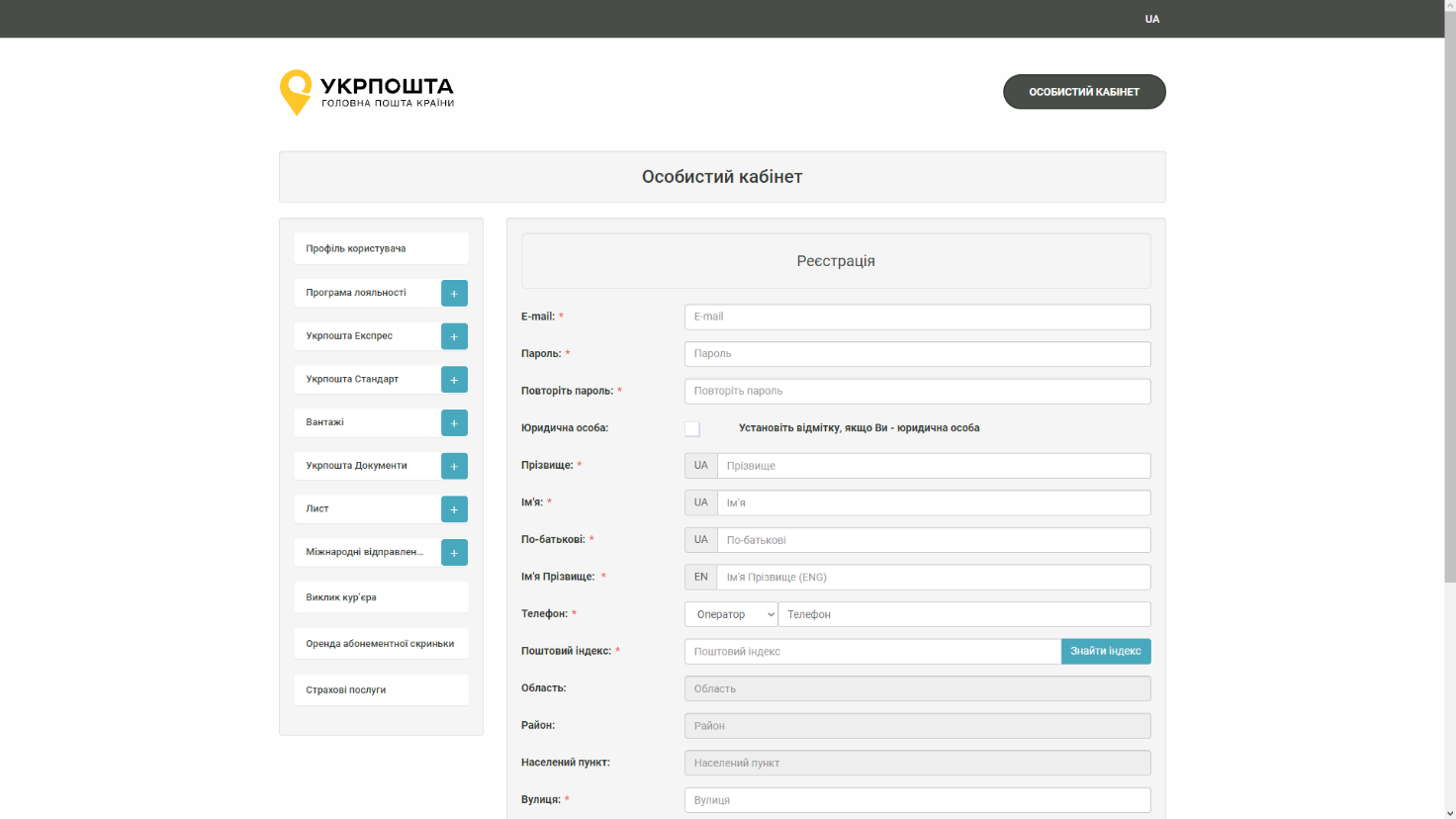


Рисунок 1.5 – Сторінка з особистим кабінетом

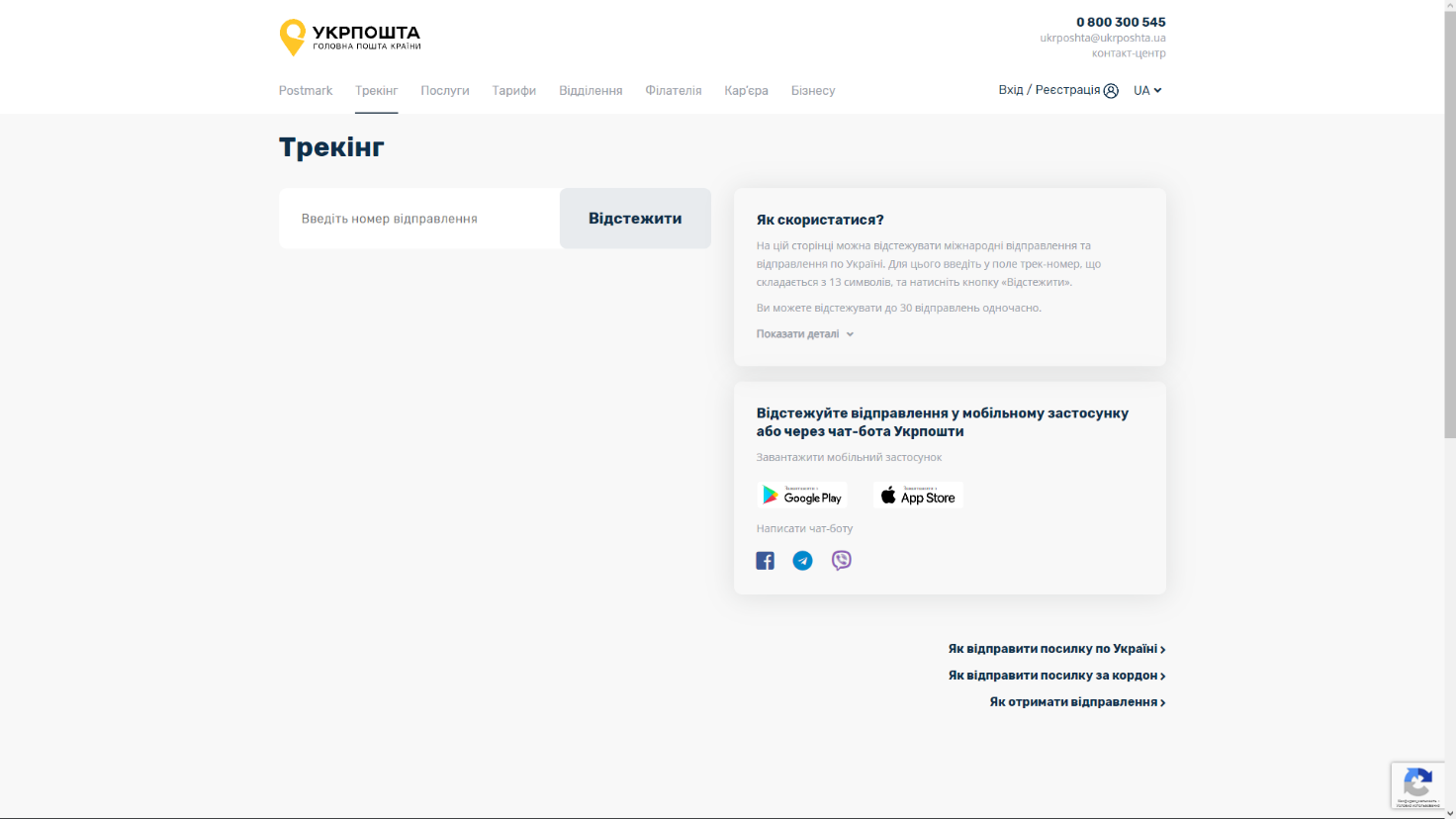


Рисунок 1.6 – Сторінка з відстеженням (трекінгом) посилки

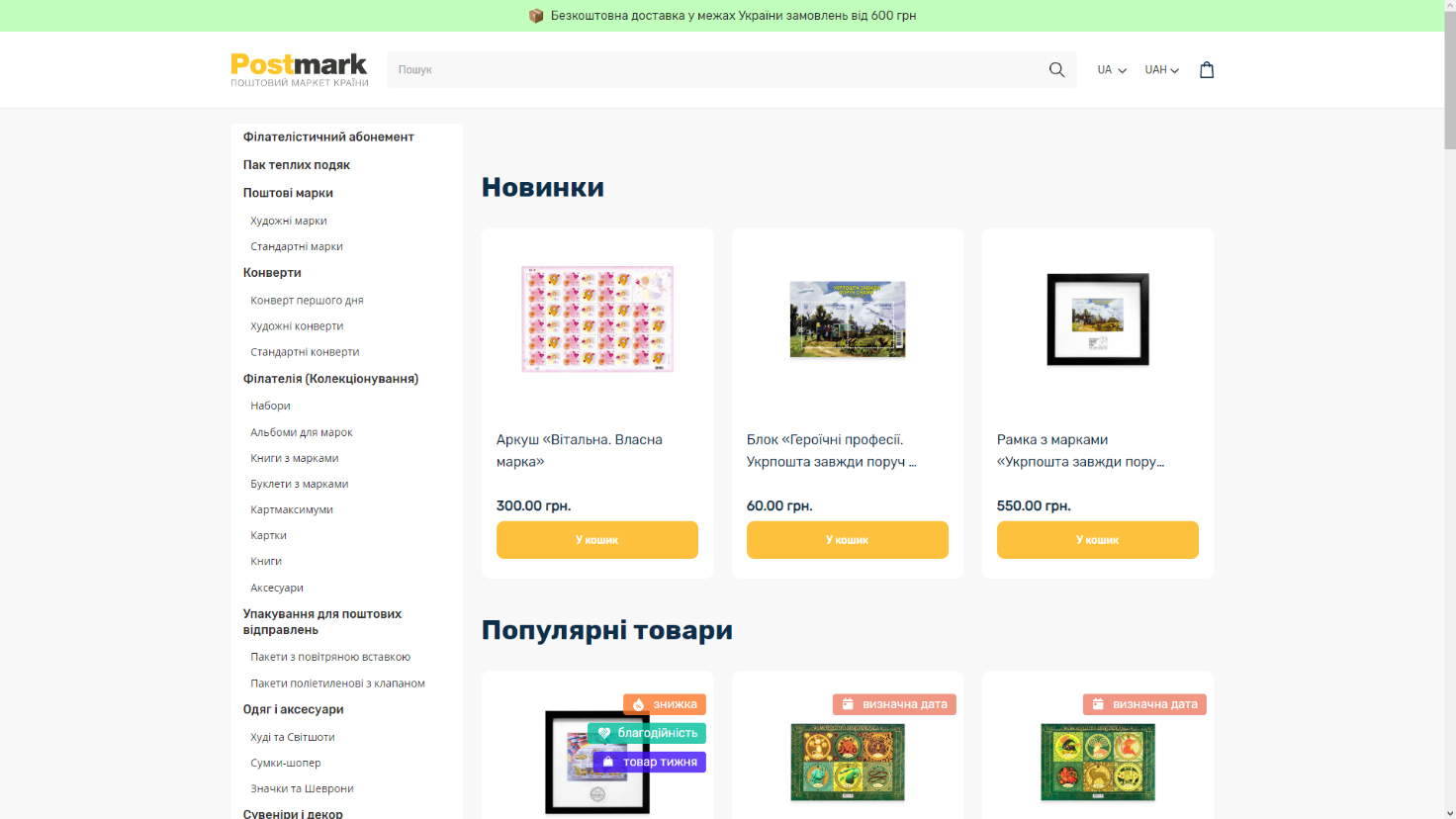


Рисунок 1.7 – Поштовий маркет сайту «Укрпошта»

## Побудова Mind Map для заданої предметної області

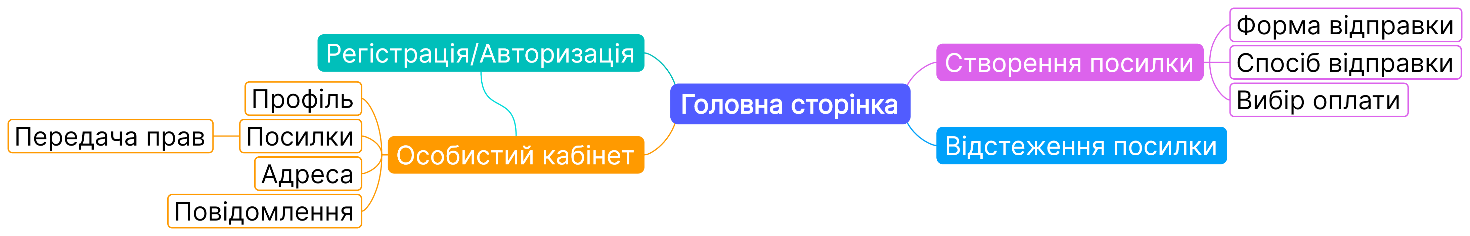


Рисунок 1.8 – Mind map

## Виділення ролей користувачів і формулювання функціональних вимог

Були виділені три ролі користувачів: «Адміністратор», «Зареєстрований користувач», «Гість».

Опишемо функціональні вимоги кожного типу користувачів.

Таблиця 1.1 – Функціональні вимоги користувача з роллю «Адміністратор»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| А.1. | Адміністратор може додавати, видаляти та редагувати користувачів.  ***Примітка:***  Адміністратор може редагувати та видаляти користувачів, але не може видаляти або редагувати особисті дані користувач без їх згоди. |
| А.2. | Адміністратор може дивитися та аналізувати дані користувачів та посилок.  ***Примітка:***  Адміністратор може переглядати статистику щодо користувачів та їх посилок для аналізу та вдосконалення роботи системи. |
| А.3. | Адміністратор може редагувати статус посилок.  ***Примітка:***  Адміністратор може змінювати статус оплати посилок. |

Таблиця 1.2 – Функціональні вимоги користувача з роллю «Зареєстрований користувач»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| ЗК.1. | Зареєстрований користувач може створювати нову посилку.  ***Примітка:***  Користувач може створити нову посилку, але не може редагувати вже відправлену посилку. |
| ЗК.2. | Зареєстрований користувач може передивлятися інформацію про статус та маршрут посилки.  ***Примітка:***  Користувач може переглядати інформацію про статус посилки в режимі реального часу. |
| ЗК.3. | Зареєстрований користувач може оплатити доставку онлайн через сайт  ***Примітка:***  Користувач може оплатити замовлення оффлайн у відділені. |
| ЗК.4. | Зареєстрований користувач може керувати особистим профілем.  ***Примітка:***   1. Користувач може змінювати особисту інформацію у власному профілі; 2. Користувач може змінювати прив’язаний номер телефона, пошту та пароль. |
| ЗК.5. | Зареєстрований користувач може передавати права на отримання посилки іншому зареєстрованому користувачу.  ***Примітка:***  Користувач може передати право на отримання посилки іншому зареєстрованому користувачу через відповідну функцію у своєму профілі. |

Таблиця 1.3 – Функціональні вимоги користувача з роллю «Гість»

|  |  |
| --- | --- |
| **Іденти-фікатор** | **Функціональні вимоги** |
| Г.1. | Гість може подивитися інформацію про послуги.  ***Примітка:***  Гість може переглядати інформацію про доступні послуги та їх ціни не зареєструвавшись. |
| Г.2. | Гість може отримати інформацію про статус доставки.  ***Примітка:***  Гість може переглядати загальну інформацію про статус доставки посилки за допомогою ідентифікаційного номера посилки. |
| Г.3. | Гість може зареєструватись на сайті.  ***Примітка:***   1. Обов’язкова прив’язка к телефону або пошти; 2. Обов’язкове введення ім’я, прізвища та по-батькові користувача; 3. Пароль повинен бути довшим за 7 символів. |

## Проектування інтерфейсу користувача

Для реалізації замовником було обрано web-застосунок.

На основі сформульованих функціональних вимог були розроблені екранні форми, які наведені на рис. 1.9 – 1.15).

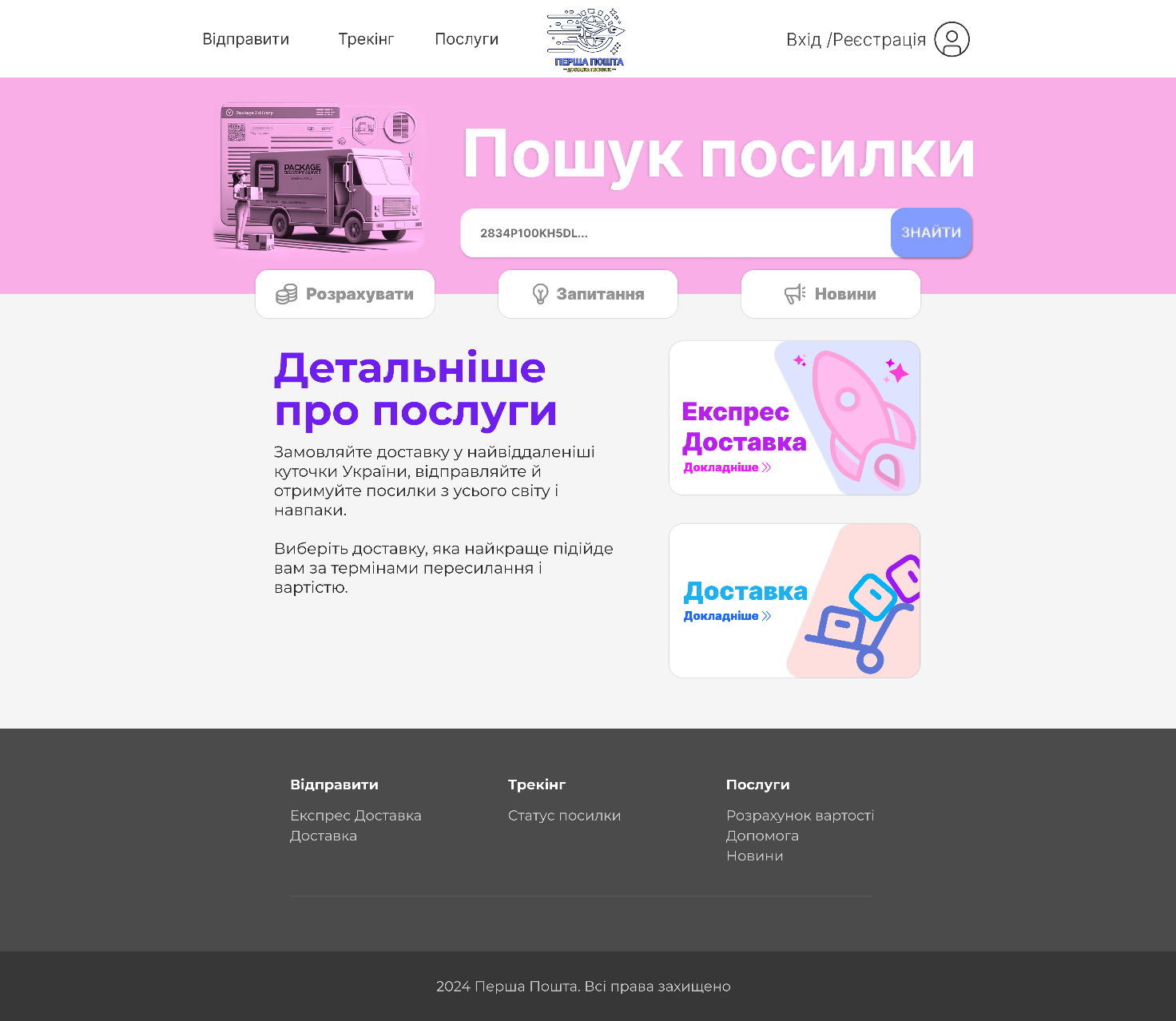


Рисунок 1.9 – Макет екранної форми web сторінки «Головна»

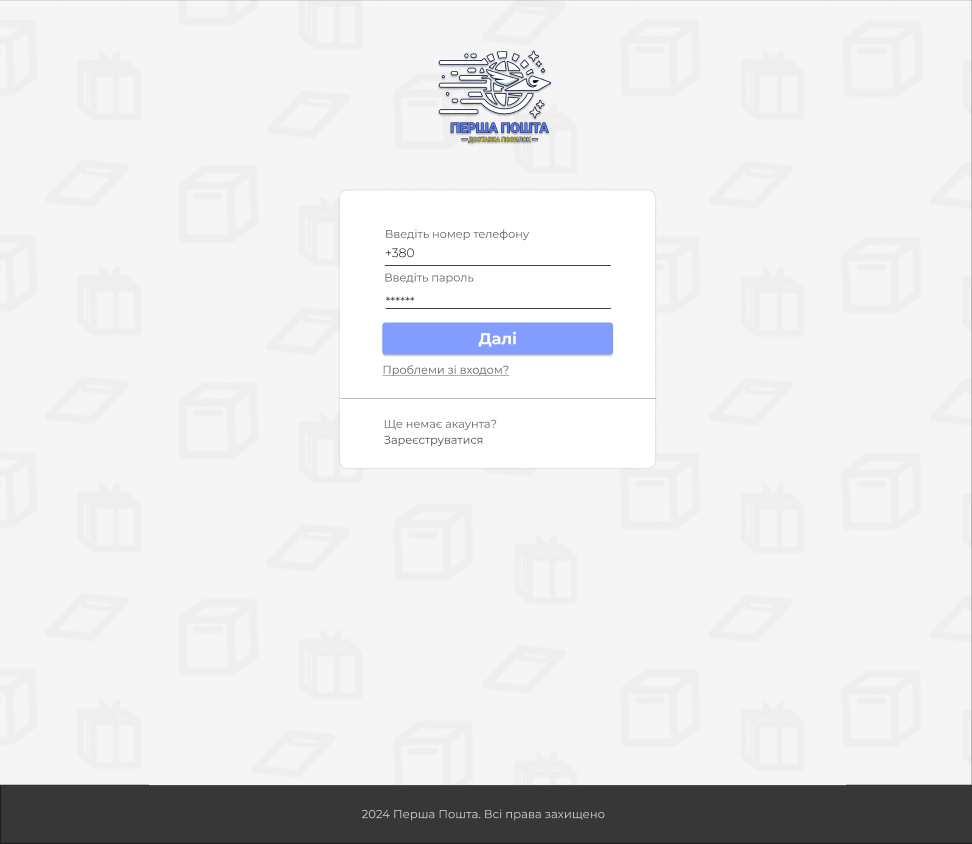
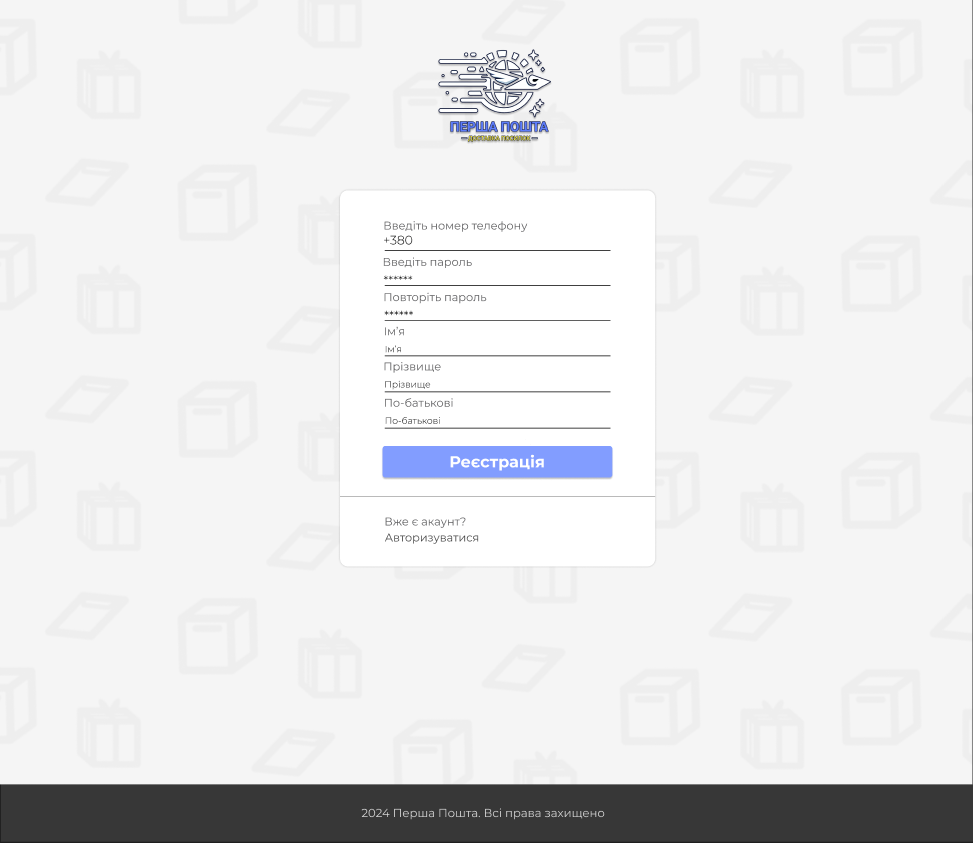


Рисунок 1.10 – Макет екранної форми web сторінки «Авторизація»

Рисунок 1.11 – Макет екранної форми web сторінки «Реєстрація»

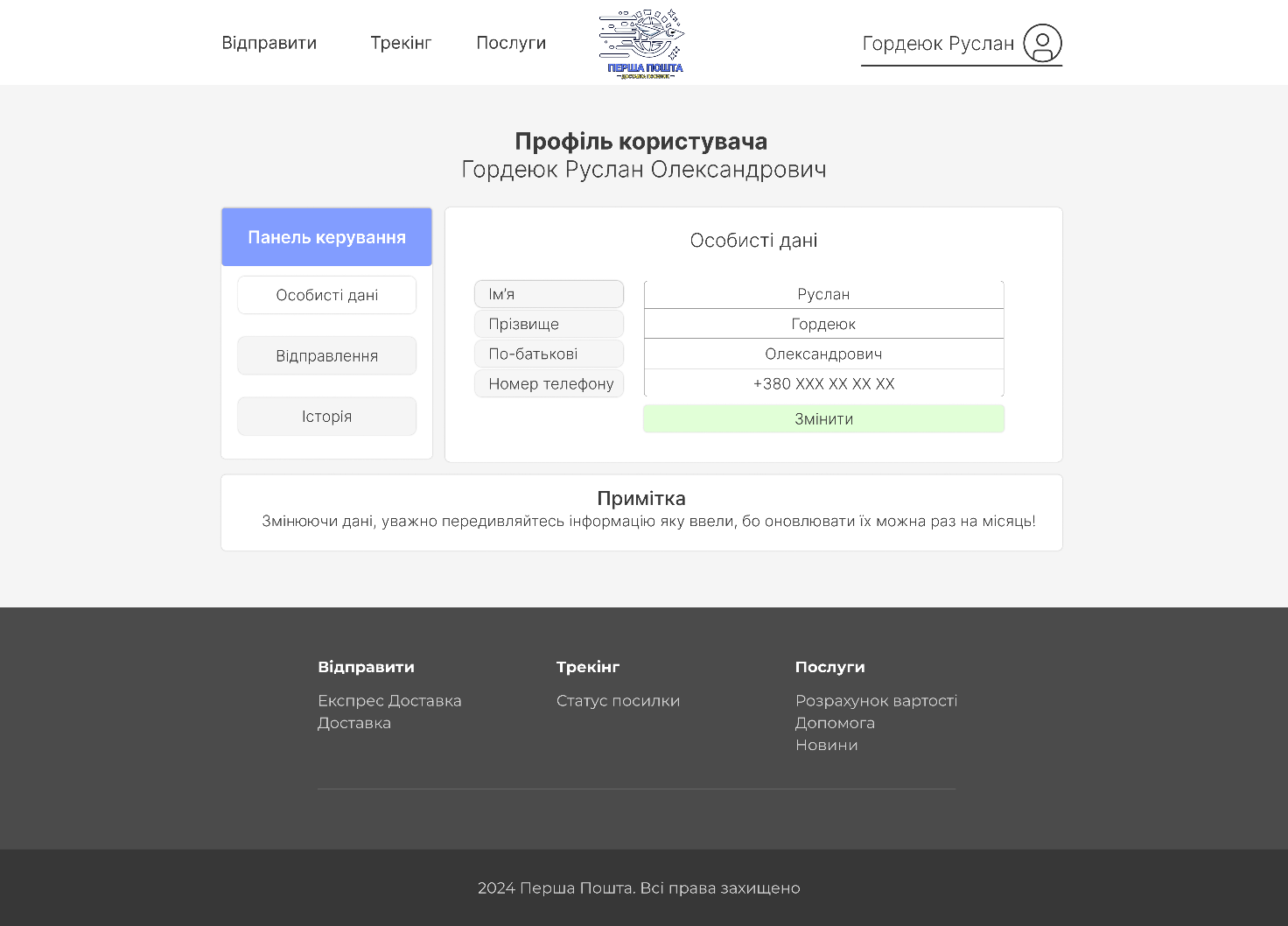


Рисунок 1.12 – Макет екранної форми web сторінки «Профіль»

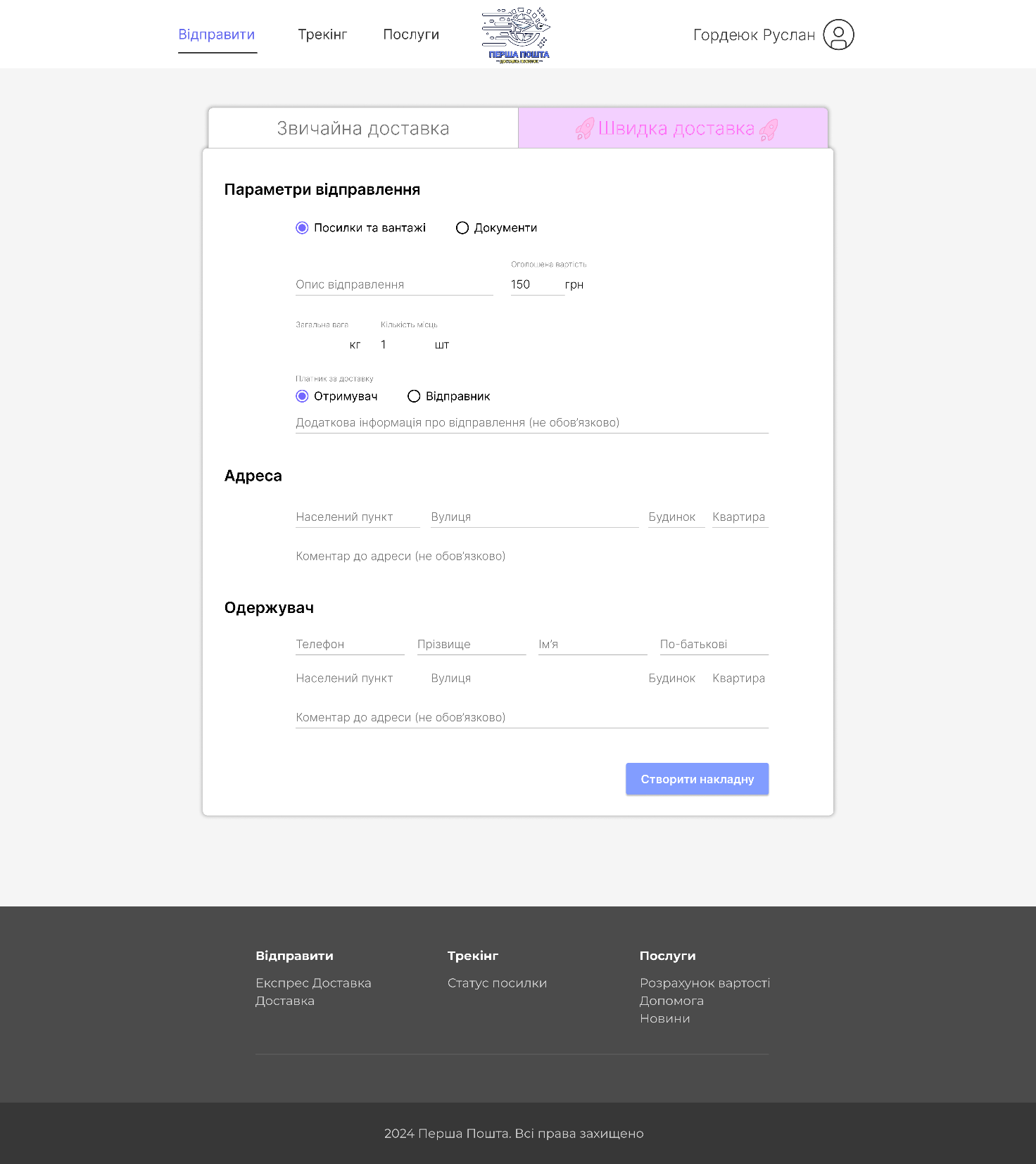


Рисунок 1.13 – Макет екранної форми web сторінки «Відправити»

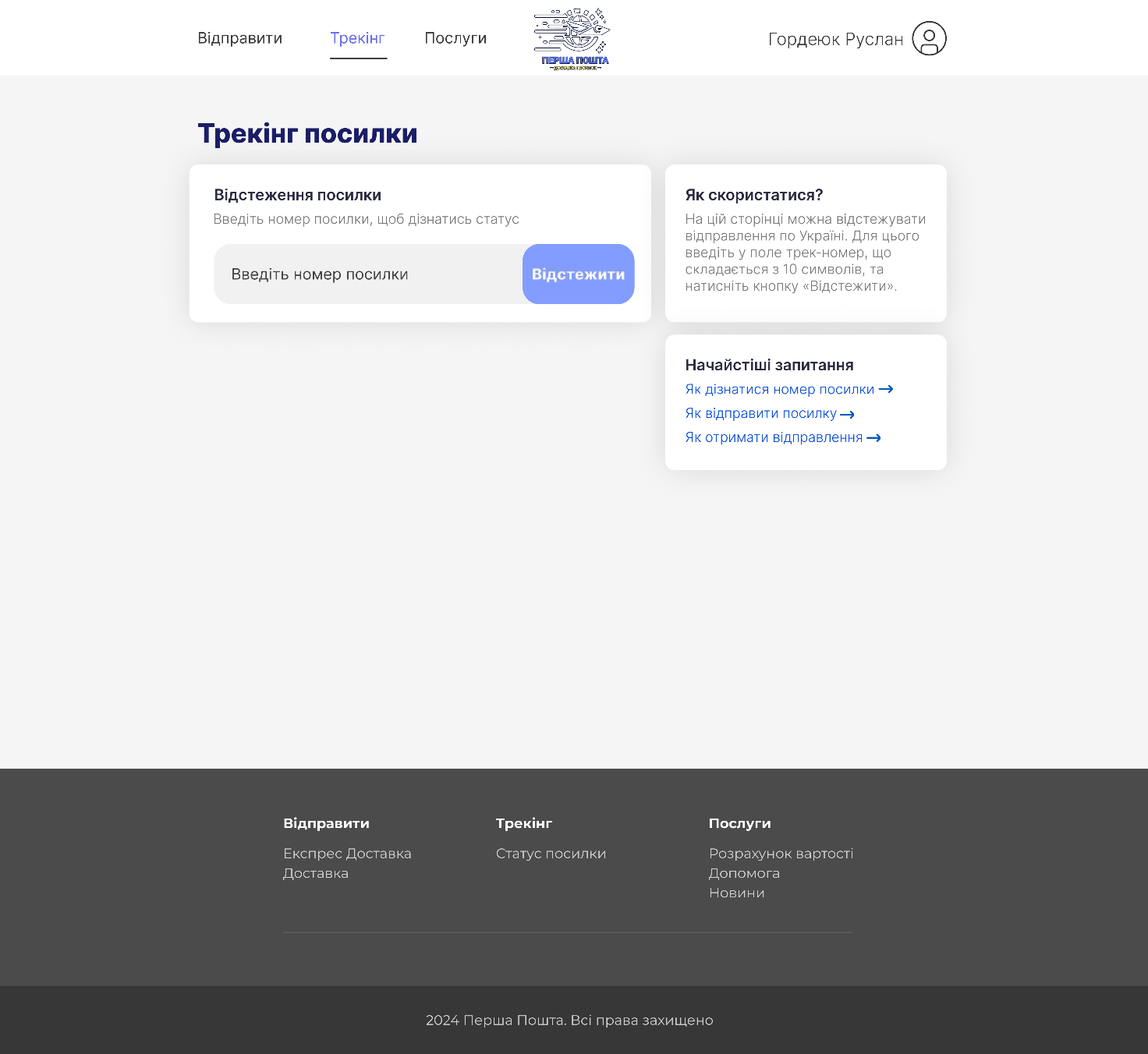


Рисунок 1.14 – Макет екранної форми web сторінки «Трекінг»

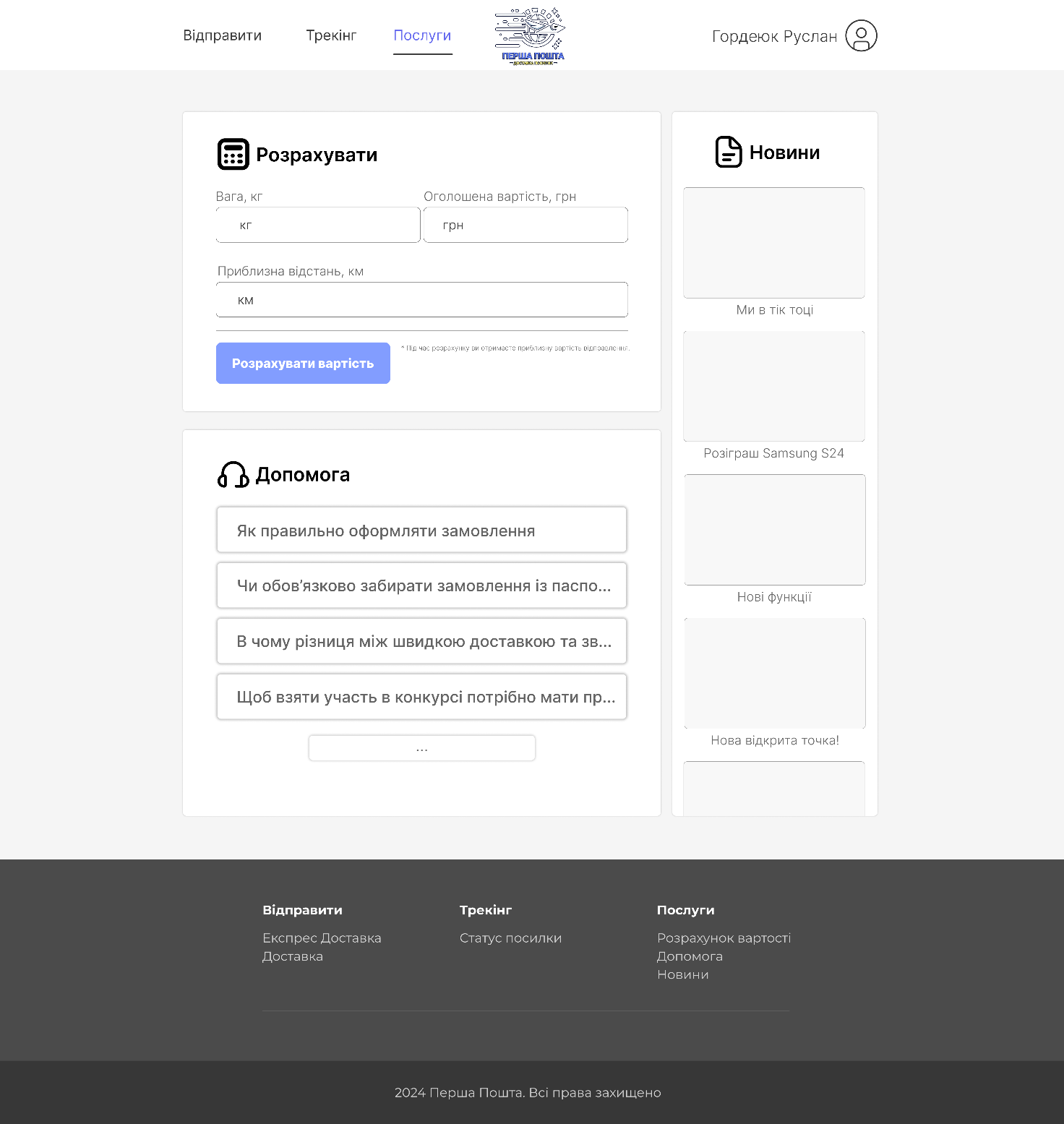


Рисунок 1.15 – Макет екранної форми web сторінки «Послуги»

## Розроблення функціональних тестів

### Функціональні тести для ролі користувача «Адміністратор»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Адміністратор» (див. табл. 1.4).

Таблиця 1.4 – Функціональні тести для ролі користувача «Адміністратор»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *А.1. Адміністратор може додати/видалити користувачів.* | | |
| А.1.1 | 1. Ввести мобільний номер: «0689999999». 2. Ввести пароль: «abcde12345». 3. Ввести ім’я: «Руслан». 4. Натиснути кнопку «Додати». | Успішне додавання користувача |
| А.1.2 | 1. Обрати користувача і поруч з ним натиснути кнопку «Редагувати». 2. Натиснути кнопку «Видалити». | Успішне видалення користувача |
| А.1.3 | 1. Ввести мобільний номер: «0689999999». 2. Ввести пароль: «пароль12345». 3. Ввести ім’я: «Руслан». 4. Натиснути кнопку «Додати». | Користувач не був доданий. Адміністратор отримав повідомлення «Дані не відповідають вимогам!» |
| *А.2. Адміністратор додати/видалити статті.* | | |
| А.2.1 | 1. Натиснути кнопку «Додати нову статтю». 2. Написати заголовок та текст статті. 3. Натиснути кнопку «Додати». | Успішне додавання статті |
| А.2.2 | 1. Обрати статтю і поруч з нею натиснути кнопку «Редагувати». 2. Натиснути кнопку «Видалити». | Успішне видалення статті |

### Функціональні тести для ролі користувача «Зареєстрований користувач»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Зареєстрований користувач» (див. табл. 1.5).

Таблиця 1.5 – Функціональні тести для ролі користувача «Зареєстрований користувач»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *ЗК.1. Зареєстрований користувач може авторизуватися.* | | |
| ЗК.1.1 | 1. Натиснути кнопку «Вхід». 2. Ввести мобільний номер: «0689999999». 3. Ввести пароль: «abcde12345». 4. Натиснути кнопку «Далі». | Успішна авторизація користувача |
| ЗК.1.2 | 1. Натиснути кнопку «Вхід». 2. Ввести мобільний номер: «0689999999». 3. Ввести пароль: «пароль12345». 4. Натиснути кнопку «Далі». | Користувач не був авторизований. Користувач отримав повідомлення «Неправильний логін або пароль!» |
| *ЗК.2 Зареєстрований користувач може створити нову посилку.* | | |
| ЗК.2.1 | 1. Натиснути кнопку «Відправити». 2. Вибрати доставку (звичайна або швидка). 3. Заповнити дані посилка. 4. Натиснути кнопку «Створити посилку» | Успішна створена накладна користувача |
| ЗК.2.2 | 1. Натиснути кнопку «Відправити». 2. Вибрати доставку (звичайна або швидка). 3. Заповнити неправильно дані накладної. 4. Натиснути кнопку «Створити посилку» | Накладна не була створена. Користувач отримав повідомлення «Неправильно введені дані!» |
| *ЗК.3 Зареєстрований користувач може передивитися статус посилки.* | | |
| ЗК.3.1 | 1. Зайти в особистий профіль натиснувши на профіль. 2. Натиснути кнопку «Посилки». | Успішно відображені історії посилок |
| *ЗК.4 Зареєстрований користувач може керувати особистим профілем.* | | |
| ЗК.4.1 | 1. Зайти в особистий профіль натиснувши на профіль. 2. Натиснути кнопку «Особисті дані». 3. Змінити дані в полі, після чого натиснути кнопку «Змінити». | Успішна зміна особистих даних |

### Функціональні тести для ролі користувача «Гість»

На основі опису предметної області, а також функціональних вимог і екранних форм були розроблені функціональні тести для ролі користувача «Гість» (див. табл. 1.6).

Таблиця 1.6 – Функціональні тести для ролі користувача «Гість»

|  |  |  |
| --- | --- | --- |
| **Іденти-фікатор**  **тесту** | **Послідовність дій користувача** | **Очікуваний результат** |
| *Г.1. Гість може розрухувати вартість доставки.* | | |
| Г.1.1 | 1. Натиснути кнопку «Послуги». 2. Вибрати послугу «Розрахунок вартості посилки». | Успішне використання послуги |
| *Г.2. Гість може подивитися статус посилки.* | | |
| Г.2.1 | 1. Натиснути кнопку «Трекінг». 2. Ввести номер існуючої посилки. 3. Натиснути кнопку «Відстежити». | Успішне виведення інформації про статус посилки |
| Г.2.2 | 1. Натиснути кнопку «Трекінг». 2. Ввести номер не існуючої посилки. 3. Натиснути кнопку «Відстежити». | Інформація не була виведена. Користувач отримав повідомлення «Не знайдено» |
| *Г.3. Гість може зареєструватися, вказавши номер телефону, пароль та ПІБ.* | | |
| Г.3.1 | 1. Ввести мобільний номер: «0689999999». 2. Ввести пароль: «abcde12345». 3. Ввести ім’я: «Руслан». 4. Натиснути кнопку «Додати». | Успішна реєстрація користувача |
| Г.3.2 | 1. Ввести мобільний номер: «+380689999999». 2. Ввести пароль: «пароль12345». 3. Ввести ім’я, прізвище, по-батькові: «Руслан». 4. Натиснути кнопку «Додати». | Користувач не був зареєстрований. Користувач отримав повідомлення «Дані не відповідають вимогам!» |

# МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

## Виділення і опис класів предметної області

**Клас «Package» – посилка**

Таблиця 2.1 – Виділення характеристик класу «Package»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Id | Ціле число | Авто запис |
| Опис | Рядок | немає |
| Тип доставки | Булеве значення | true, false |
| Вага | Ціле число | Не пустий рядок, довжина: хоча б 1 символ |
| Ціна | Ціле число | Не пустий рядок, довжина: хоча б 1 символ |
| Ім’я відправника | Рядок | немає |
| Ім’я отримувача | Рядок | немає |
| Номер телефону отримувача | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Адреса отримувача | Enum | Потрібно вибрати список міст |
| Id відправника | Рядок | немає |
| Id отримувача | Рядок | немає |

Таблиця 2.2 – Виділення поведінки класу «Package»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Бізнес-правила** |
| Збереження в БД | Клас Package | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то зберегти посилку в БД. |

**Клас «AccModel» – користувач**

Таблиця 2.3 – Виділення характеристик класу «AccModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Id | Ціле число | Авто запис |
| Ім’я | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Пошта | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Номер телефону | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Роль користувача | Enum | «User», «Admin» |
| Місто | Enum | Тільки міста які є в enum |
| Пароль | Рядок | Не пустий рядок, який має формат: хоча 6 символів |

Таблиця 2.4 – Виділення поведінки класу «AccModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Збереження в БД | Клас AccModel | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то зберегти нового користувача в БД. |

**Клас «AddUserModel» – додати користувача**

Таблиця 2.5 – Виділення характеристик класу «AddUserModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Ім’я | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Пошта | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Номер телефону | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Пароль | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Роль | Enum | «User», «Admin» |
| Місто | Enum | Тільки міста які є в enum |

Таблиця 2.6 – Виділення поведінки класу «AddUserModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Додання користувача через панель адміністратора | Клас AddUserModel | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то зберегти нового користувача в БД. |

**Клас «ArticleModel» – стаття**

Таблиця 2.7 – Виділення характеристик класу «ArticleModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| ID | Ціле число | > 0 |
| Заголовок | Рядок | Не пустий рядок, довжина: до 50 символів |
| Опис | Рядок | Не пустий рядок, довжина: хоча б 1 символ |

Таблиця 2.8 – Виділення поведінки класу «ArticleModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Збереження в БД | Клас ArticleModel | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то зберегти статтю в БД. |
| Виведення статті | - | Клас ArticleModel | 1. Виведення всіх існуючих статей починаючи з перщої |

**Клас «CalcModel» – калькулятор**

Таблиця 2.9 – Виділення характеристик класу «CalcModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Вага | Число з плаваючою комою | >0.0 |
| Ціна | Число з плаваючою комою | >0.0 |
| Місто | Enum | Тільки міста які є в enum |
| Вартість | Не пустий рядок, довжина: хоча б 1 символ | >0.0 |

Таблиця 2.10 – Виділення поведінки класу «CalcModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Розрахунок приблизної вартості доставки | Клас CalcModel | Double – число з плаваючою комою | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то розрахувати вартість доставки. |

**Клас «LoginViewModel» – авторизація**

Таблиця 2.11 – Виділення характеристик класу «LoginViewModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Номер телефону | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Пароль | Рядок | Не пустий рядок, довжина: хоча б 6 символів |
| Запам’ятати користувача | Булеве значення | true, false |

Таблиця 3.2 – Виділення поведінки класу «LoginViewModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Авторизація користувача | Клас LoginViewmodel | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то ввійти в систему. |

**Клас «RegisterViewModel» – реєстрація**

Таблиця 2.12 – Виділення характеристик класу «RegisterViewModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Ім’я | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Пошта | Рядок | Не пустий рядок, має відповідати формату: user@mail |
| Номер телефону | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Пароль | Рядок | Не пустий рядок, довжина: хоча б 6 символів |
| Підтвердити пароль | Рядок | Не пустий рядок, довжина: хоча б 6 символів |

Таблиця 2.13 – Виділення поведінки класу «RegisterViewModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Регістрація користувача | Клас LoginViewmodel | - | 1. Перевірити коректність даних. 2. Якщо пройдені перевірки, то додати користувача в БД. |

**Клас «TrackingModel» – трекінг посилки**

Таблиця 2.14 – Виділення характеристик класу «TrackingModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Id | Ціле число | >0 |
| Знайдено | Булеве значення | true, false |
| Текст | Рядок | немає |
| Ім’я | Рядок | немає |
| Опис | Рядок | немає |
| Місто | Рядок | немає |

Таблиця 2.15 – Виділення поведінки класу «TrackingModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Пошук посилки | Ціле число | Клас TrackingModel | 1. Перевіряється наявність посилки. 2. Якщо пройдені перевірки, то повертається інформація про посилку. |

**Клас «AdminPanelViewModel» – панель адміна**

Таблиця 2.16 – Виділення характеристик класу «AdminPanelViewModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Аккаунти користувачів | Список класів | немає |
| Список статті | Список класів | немає |

Таблиця 2.17 – Виділення поведінки класу «AdminPanelViewModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Виведення інформації | - | Клас AdminPanelViewModel | - |

**Клас «ChangePasswordViewModel» – змінити пароль**

Таблиця 2.18 – Виділення характеристик класу «ChangePasswordViewModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Пароль | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Новий пароль | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Підтвердити пароль | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Змінений | Булеве значення | true, false |

Таблиця 2.19 – Виділення поведінки класу «ChangePasswordViewModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Зміна паролю | Пароль, Новий пароль, Підтвердити пароль | Змінений - bool | 1. Перевіряється коректність даних. 2. Якщо пройдені перевірки, то змінюється пароль. |

**Клас «ViewProfileViewModel» – профіль**

Таблиця 2.20 – Виділення характеристик класу «ViewProfileViewModel»

|  |  |  |
| --- | --- | --- |
| **Назва характеристики** | **Тип значення** | **Обмеження** |
| Ім’я | Рядок | Не пустий рядок, довжина: хоча б 1 символ |
| Номер телефону | Рядок | Не пустий рядок, який має формат: (XXX)-XXX-XX-XX |
| Місто | Enum | Тільки міста які є в enum |
| Змінений | Рядок | true, false |

Таблиця .21 – Виділення поведінки класу «ViewProfileViewModel»

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва поведінки** | **Опис вхідних параметрів** | **Тип значення, що повертається** | **Примітка** |
| Пошук посилки | Ціле число | Клас TrackingModel | 1. Перевіряється наявність посилки. 2. Якщо пройдені перевірки, то повертаєтья інформація про посилку. |

## Встановлення зв’язків між класами

На рисунку 2.1 наведена діаграма класів, яка була побудова з використанням сервісу <https://www.lucidchart.com>.

|  |  |
| --- | --- |
|  | Рисунок 2.1 – Діаграма класів |

# ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ПРЕДМЕТНОЇ ОБЛАСТІ І ЇХ ТЕСТУВАННЯ

## Структура проекту з реалізацією класів предметної області

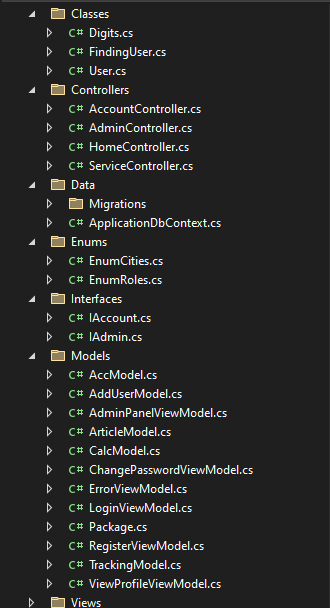


Рисунок 3.1 – Структура проекту з реалізацією предметної області

## Реалізація інтерфейсів і каркасів класів предметної області

**Інтерфейс «IAccount» – користувач**

В інтерфейсі «IAccount» містяться інтерфейси методів ASP.Net Core, які виконується у методі контролера «AccountController». Коді інтерфейсу наведений у [Додатку Б](#_ДОДАТОК_Б._Лістинг_1).

**Інтерфейс «IAdmin» – користувач**

В інтерфейсі «IAdmin» містяться інтерфейси методів ASP.Net Core, які виконується у методі контролера «AdminController». Код інтерфейсу наведений у [Додатку Б](#_ДОДАТОК_Б._Лістинг_1).

**Клас «AccModel» – користувач**

В класі «AccModel» містяться дані аккаунта, які потім обчислюються і використовуються в контролерах проекту. Код класу наведений у [Додатку Б](#_ДОДАТОК_Б._Лістинг_1).

**Клас «Package» – посилка**

В класі «Package» містяться дані посилки, які потім обчислюються і використовуються в контролерах проекту. Код класу наведений у [Додатку Б](#_ДОДАТОК_Б._Лістинг_1).

**Клас «ArticleModel» – посилка**

В класі «ArticleModel» містяться дані статті, з якими ми взаємодіємо в контролерах проекту. Код класу наведений у [Додатку Б](#_ДОДАТОК_Б._Лістинг_1).

## Розроблення unit-тестів для класів предметної області

Розроблені unit-тести наведені у [Додатку В](#_ДОДАТОК_В._Лістинг).

Таблиця 3.1 – Покриття unit-тестами класу «AccModel»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| AccModel | 1 |
| AccLoginModel | 1 |
| AccRegisterModel | 1 |

Таблиця 3.2 – Покриття unit-тестами класу «Package»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| PackageModel | 1 |
| PackageTrackingModel | 1 |

Таблиця 3.3 – Покриття unit-тестами класу «ArticleModel»

|  |  |
| --- | --- |
| **Назва методу** | **Кількість розроблених unit-тестів** |
| ArticleModel | 1 |

## Повна реалізація класів предметної області

У [Додатку A](#_ДОДАТОК_А._Лістинг) наведено повну реалізацію програмного коду розроблених класів предметної області.

## Результати unit-тестування класів предметної області

На рисунках 3.2 – 3.4 наведено скріншоти проходження розроблених unit-тестів для класів предметної області.



Рисунок 3.2 – Скріншот проходження unit-тестів для класу AccModel

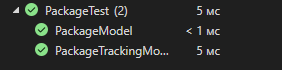


Рисунок 3.3 – Скріншот проходження unit-тестів для класу Package



Рисунок 3.3 – Скріншот проходження unit-тестів для класу ArticleModel

# ПРОЄКТУВАННЯ І ПРОГРАМНА РЕАЛІЗАЦІЯ КЛАСІВ ІНТЕРФЕЙСУ КОРИСТУВАЧА

## Структура проекту з реалізацією класів інтерфейсу користувача

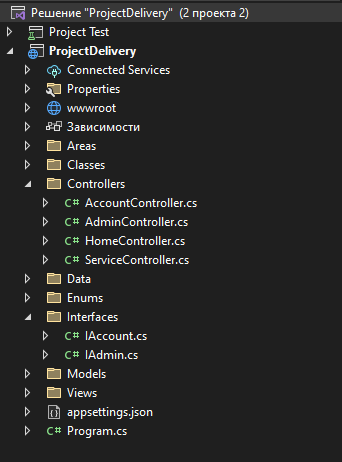


Рисунок 4.1 – Структура проекту з інтерфейсом користувача

## Виділення класів для реалізації інтерфейсу користувача

Класи HomeController, ServiceController, AdminController, AccountController успадковують базовий клас Controller. Це дає можливість використовувати всі функціональні можливості, які надає Controller.

Методи HomeController:

1. Index(): Цей метод є action методом, який обробляє GET-запити до сторінки "Index". Повертає відповідь типу IActionResult, в даному випадку - результат відображення представлення.
2. Index(TrackingModel trckmd): Цей метод є action методом, який обробляє POST-запити до сторінки "Index". Приймає об'єкт типу TrackingModel з форми, виконує перевірку валідності моделі, і, якщо модель є валідною, перенаправляє користувача на іншу сторінку "Tracking" контролера "Service".
3. Error(): Цей метод обробляє помилки і відображає сторінку з помилкою. Використовує ResponseCache атрибут, щоб заборонити кешування відповіді.

Методи ServiceController:

1. Index(): Цей метод відображає основну сторінку служби.
2. Tracking(int id): Метод для відстеження відправлення за його ідентифікатором.
3. Calculate(int w, int c, EnumCities city): Метод для обчислення вартості доставки.
4. Create(): Метод для відображення сторінки створення нового відправлення.
5. Help(): Метод для відображення списку статей.

Методи AdminController:

1. AdminPanel(): Цей метод відповідає на GET-запити і відображає адміністративну панель. Перевіряє, чи ввійшов користувач з роллю адміністратора, та передає необхідні дані в представлення.
2. AddUser(): Цей метод відображає сторінку додавання нового користувача.
3. AddUser(AddUserModel newUser): Цей метод обробляє POST-запити для додавання нового користувача. Перевіряє валідність моделі та додавання нового користувача в базу даних.
4. UpdateUser(string userId, string Name, string Email, string Phone, EnumRoles Role): Цей метод оновлює інформацію про користувача в базі даних.
5. DeleteUser(string userId): Цей метод видаляє користувача з бази даних.
6. AddArticle(): Цей метод відображає сторінку додавання нової статті.
7. AddArticle(ArticleModel newArticle): Цей метод обробляє POST-запити для додавання нової статті в базу даних.
8. UpdateArticle(int articleId, string Title, string Description): Цей метод оновлює інформацію про статтю в базі даних.
9. DeleteArticle(int articleId): Цей метод видаляє статтю з бази даних.

В AccountController визначено подію UserEventOccurred, яка відбувається при реєстрації користувача чи авторизації. В обробнику цієї події встановлюється повідомлення, яке потім відображається користувачеві.

Методи AccountController:

1. Register(): GET метод, який відображає сторінку реєстрації.
2. Register(RegisterViewModel model): POST метод, який обробляє дані, введені користувачем на сторінці реєстрації. Валідує модель та, якщо дані є вірними, створює нового користувача.
3. Login(string returnUrl = null): GET метод, який відображає сторінку входу користувача.
4. Login(LoginViewModel model, string returnUrl = null): POST метод, який обробляє дані, введені користувачем на сторінці входу. Після успішної авторизації перенаправляє користувача на попередню сторінку або на головну сторінку.
5. Logout(): POST метод, який відключає користувача від системи.
6. ChangePassword(): GET метод, який відображає сторінку зміни пароля користувача.
7. ChangePassword(ChangePasswordViewModel model): POST метод, який обробляє дані, введені користувачем на сторінці зміни пароля. Перевіряє введений поточний пароль, змінює пароль на новий та оновлює дані користувача.
8. ViewProfile(): GET метод, який відображає сторінку перегляду профілю користувача.
9. ViewProfile(ViewProfileViewModel model): POST метод, який обробляє дані, введені користувачем на сторінці перегляду профілю. Оновлює дані профілю користувача.
10. HistoryPackages(): GET метод, який відображає історію відправлень користувача.

## Програмна реалізація класів інтерфейсу користувача

У [Додатку Г](#_ДОДАТОК_Г._Лістинг) наведено повну реалізацію програмного коду розроблених класів інтерфейсу користувача.

# ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Функціональне тестування для ролі користувача «Адміністратор»

Для проведення функціонального тестування для ролі «Адміністратор» були використані розроблені у п. 1.4.1. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:** А.1.1.

**Вхідні дані:**

1. Ввести ім’я: «Користувач».
2. Ввести пошту: «userone@gmail.com».
3. Ввести мобільний номер: «0689999999».
4. Вибрати місто: «Kharkiv»
5. Вибрати роль користувача: «User»
6. Ввести пароль: «abcde12345».
7. Натиснути кнопку «Додати».

**Очікуваний результат:** Успішне додавання користувача

**Отриманий результат:**

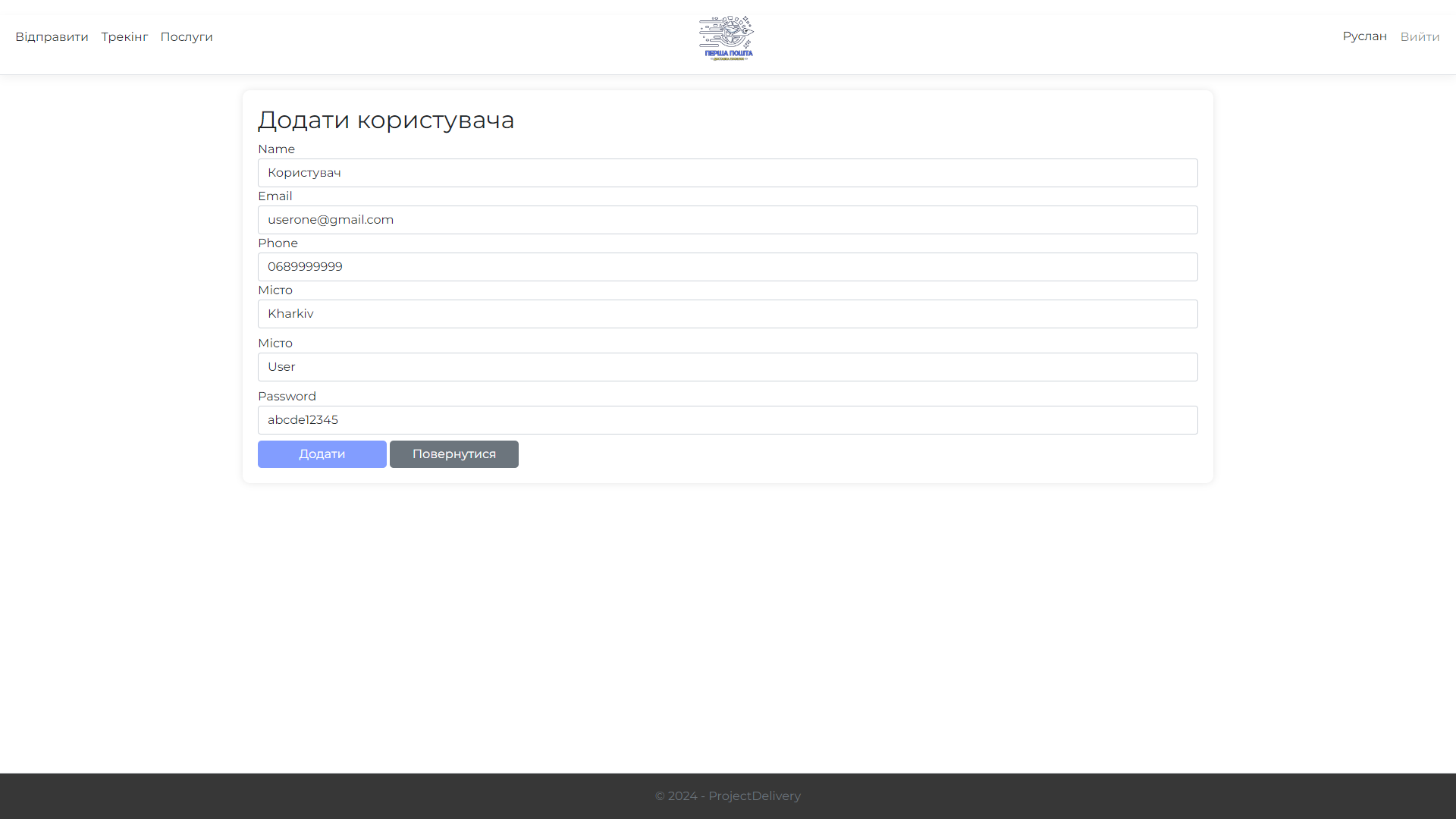


Рисунок 5.1 – Скріншот виконання тесту А.1.1

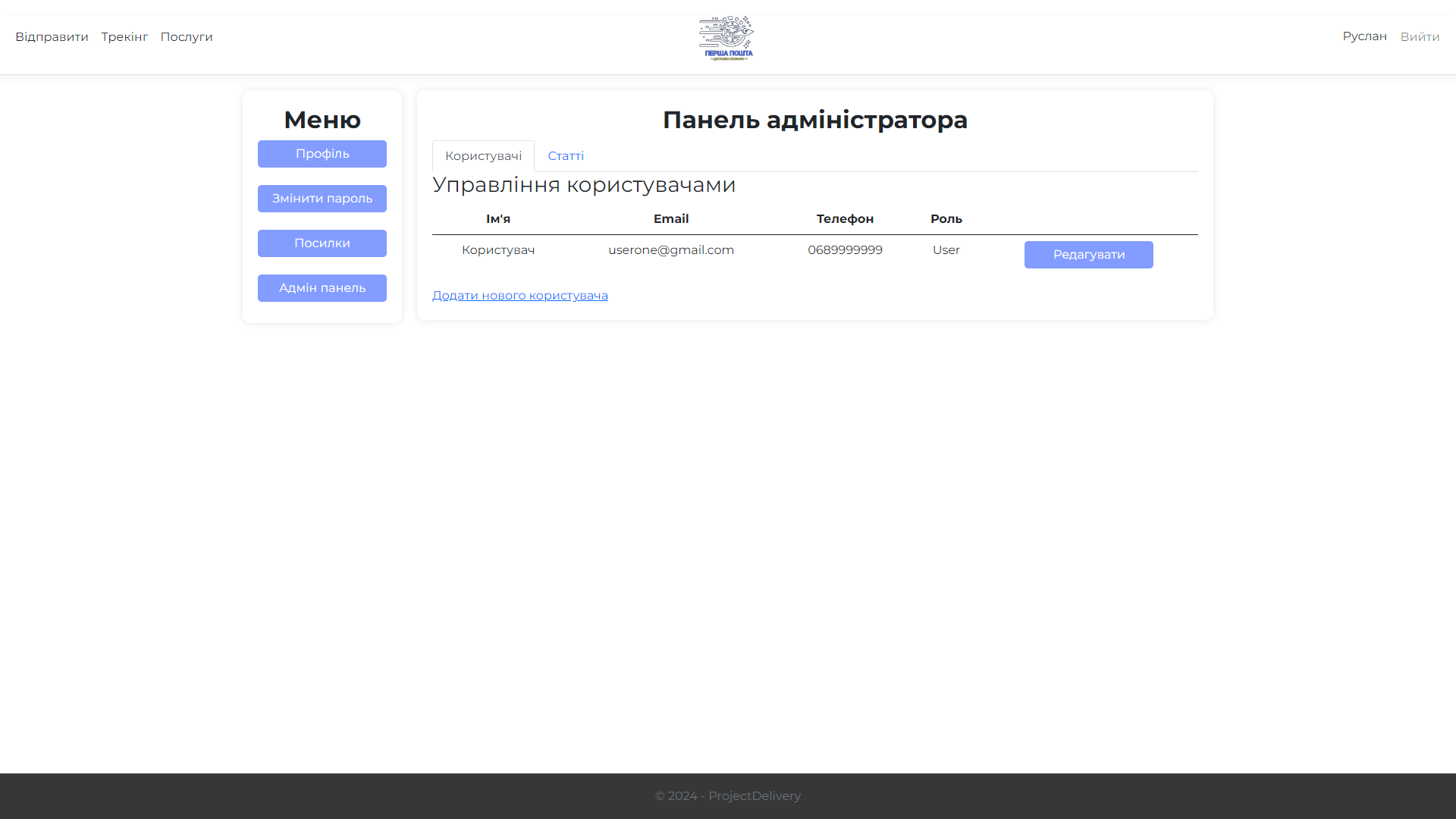


Рисунок 5.2 – Скріншот виконання тесту А.1.1

**Ідентифікатор тесту:** А.1.2.

**Вхідні дані:**

1. Обрати користувача та поряд з ним натиснути «Редагувати».
2. Натиснути кнопку «Видалити».

**Очікуваний результат:** Успішне видалення користувача

**Отриманий результат:**

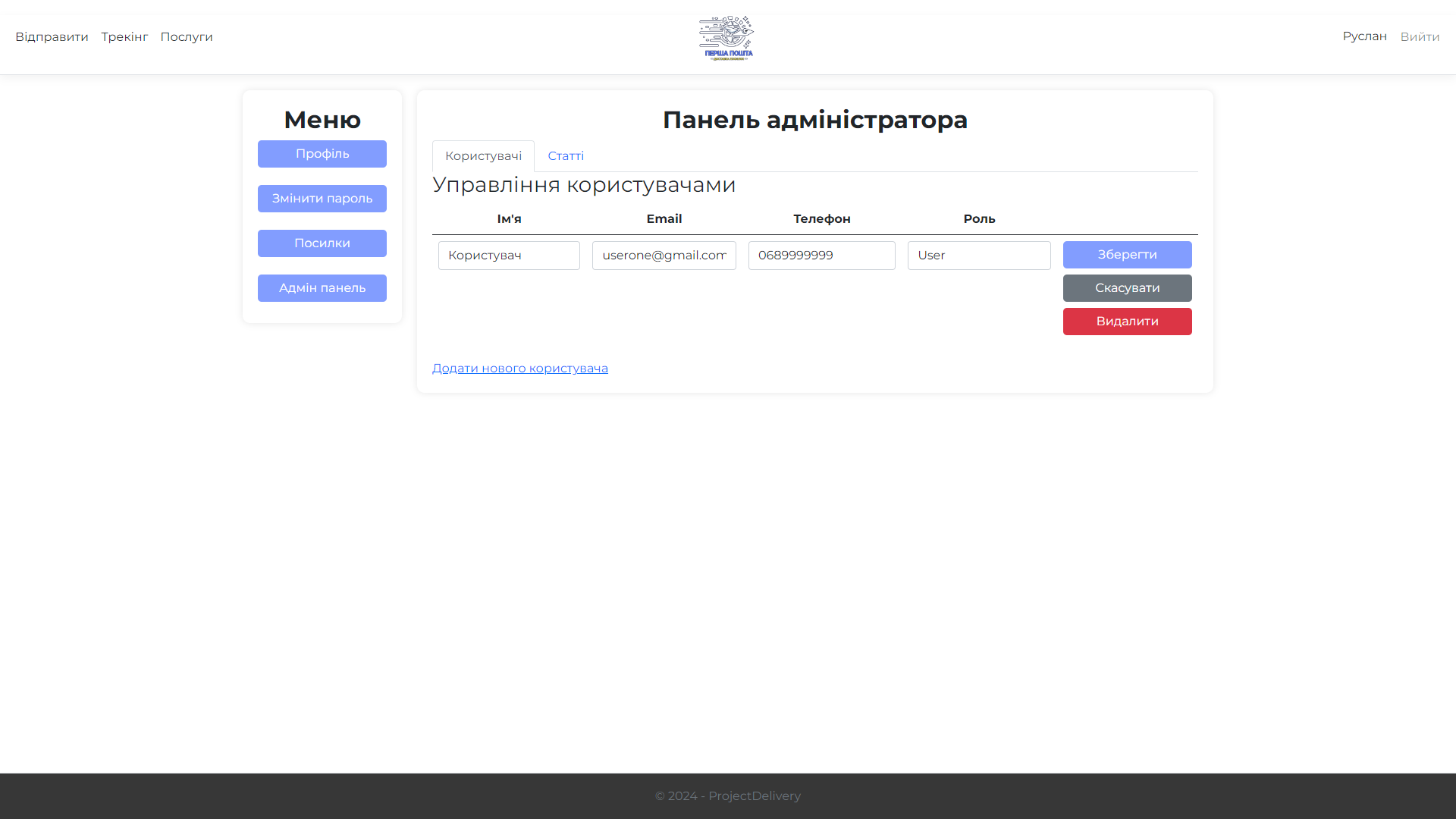


Рисунок 5.3 – Скріншот виконання тесту А.1.2

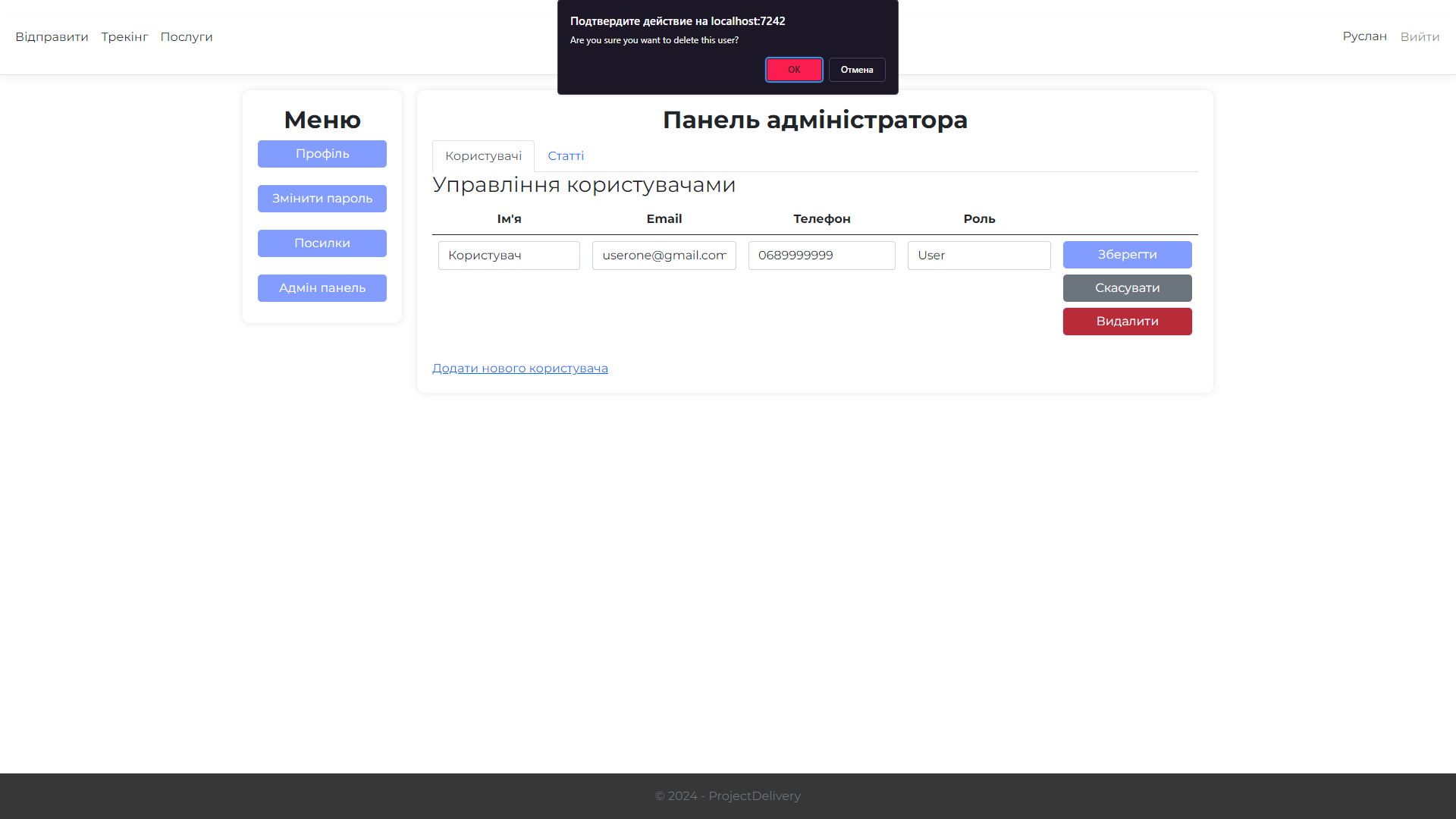


Рисунок 5.4 – Скріншот виконання тесту А.1.2

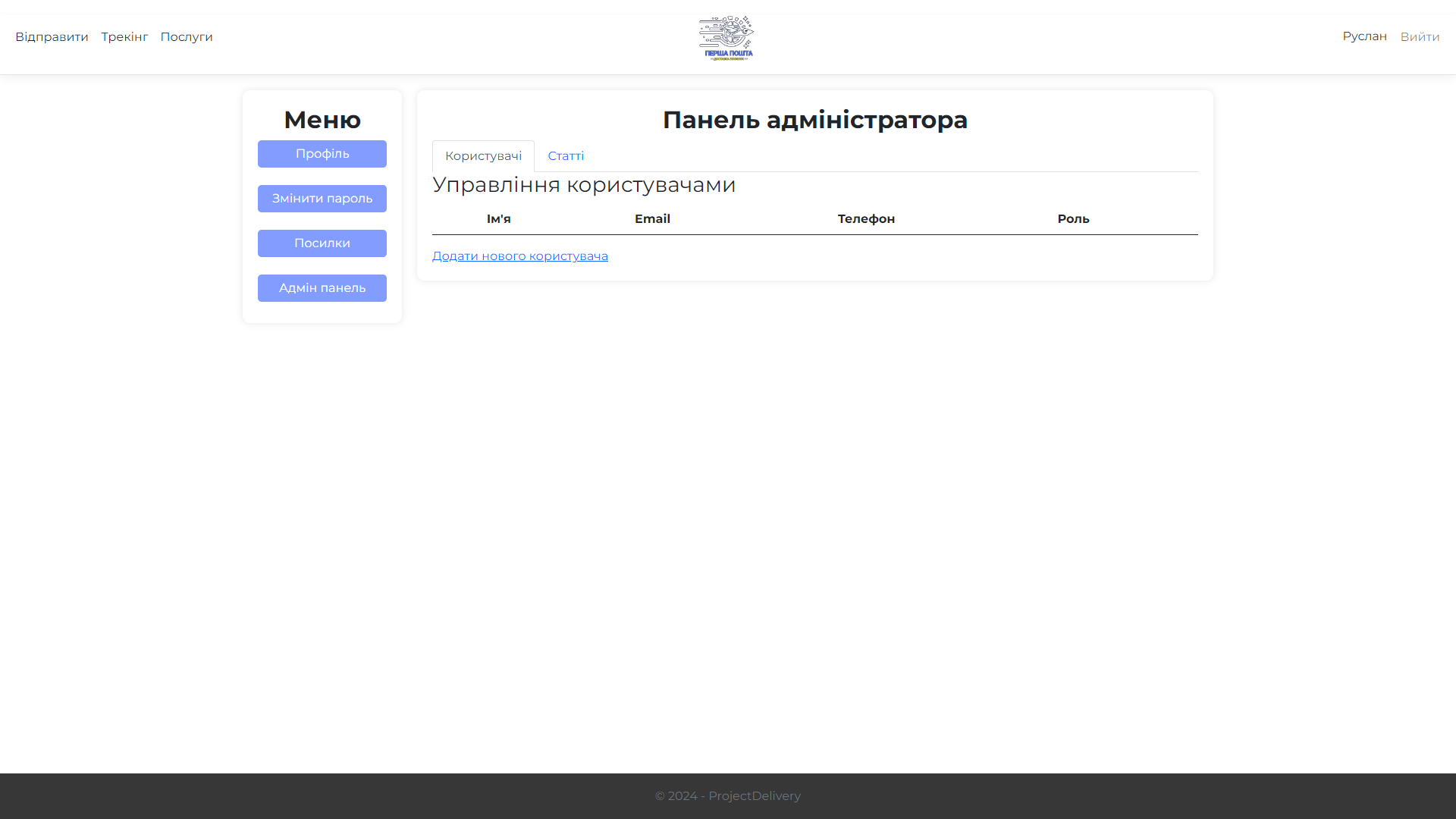


Рисунок 5.5 – Скріншот виконання тесту А.1.2

**Ідентифікатор тесту:** А.1.3.

**Вхідні дані:**

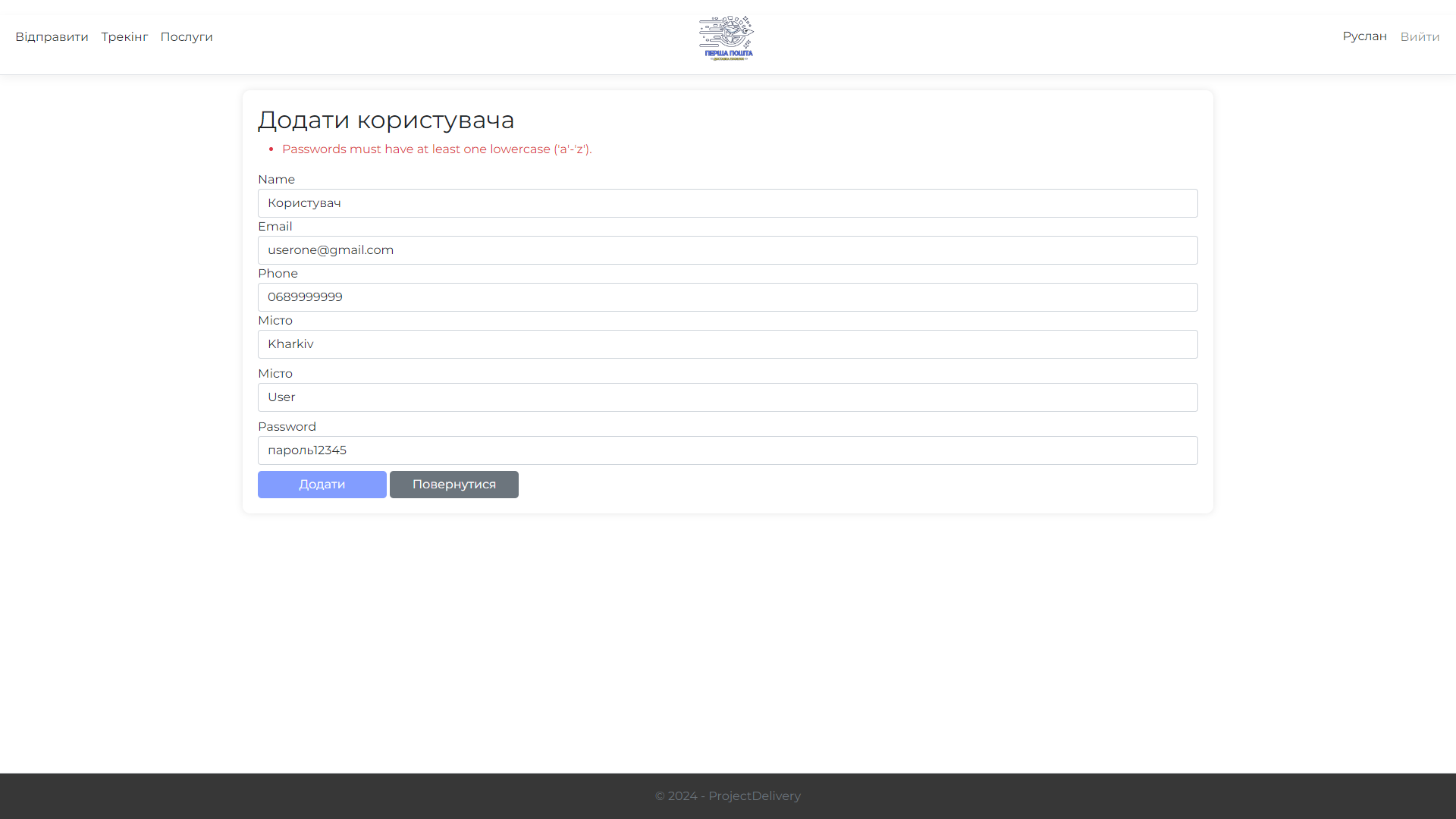
1. Ввести ім’я: «Користувач».
2. Ввести пошту: «userone@gmail.com».
3. Ввести мобільний номер: «0689999999».
4. Вибрати місто: «Kharkiv»
5. Вибрати роль користувача: «User»
6. Ввести пароль: «пароль12345».
7. Натиснути кнопку «Додати».

**Очікуваний результат:** Користувач не був доданий

**Отриманий результат:**



Рисунок 5.6 – Скріншот виконання тесту А.1.3

  
Рисунок 5.7 – Скріншот виконання тесту А.1.3

## Функціональне тестування для ролі користувача «Зареєстрований користувач»

Для проведення функціонального тестування для ролі «Зареєстрований користувач» були використані розроблені у п. 1.4.2. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:** ЗК.1.1.

**Вхідні дані:**

1. Ввести мобільний номер: «0689999999».
2. Ввести пароль.
3. Натиснути кнопку : «Ввійти».

**Очікуваний результат:** Успішний вхід користувача в систему

**Отриманий результат:**

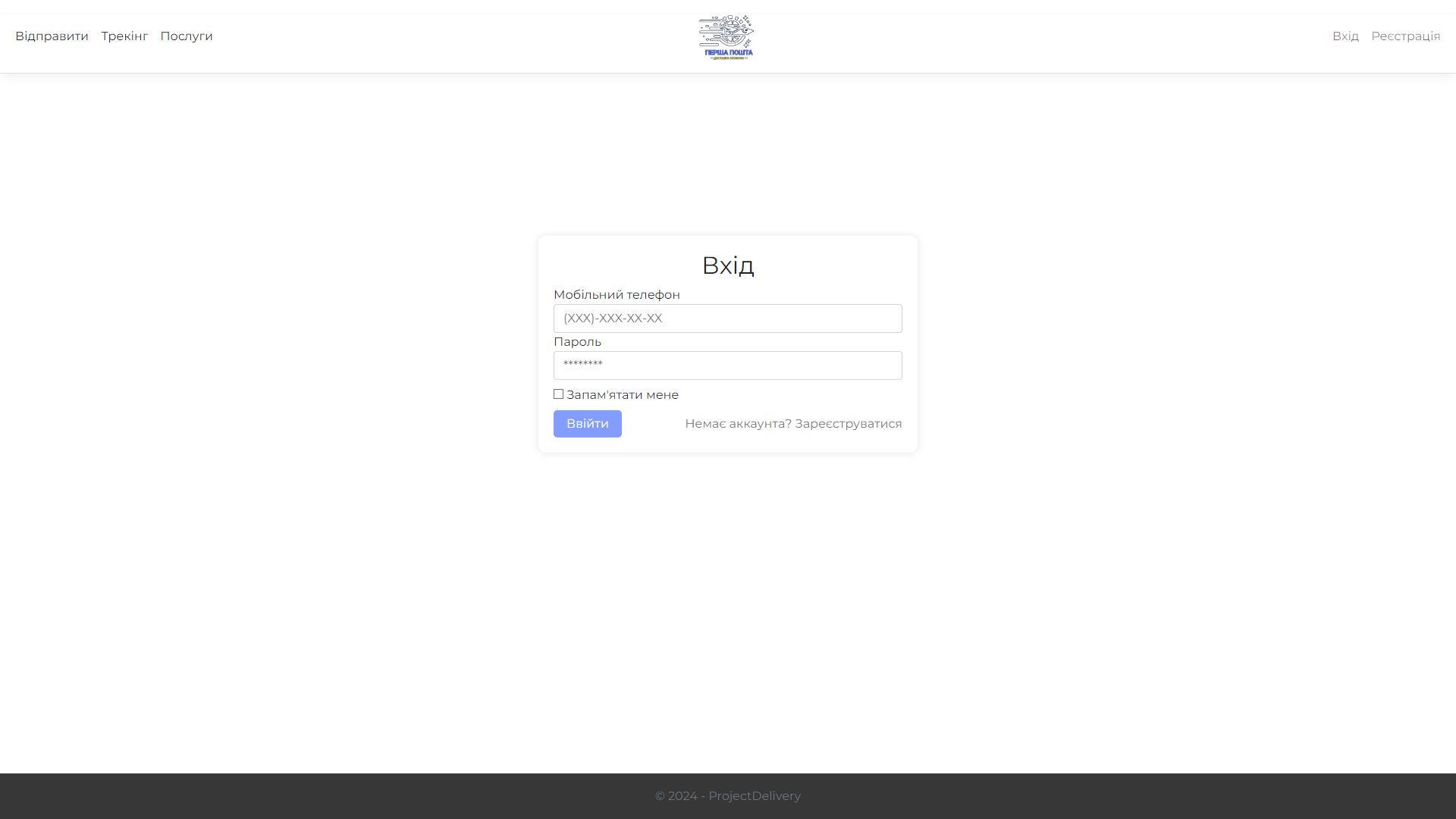


Рисунок 5.8 – Скріншот виконання тесту ЗК.1.1

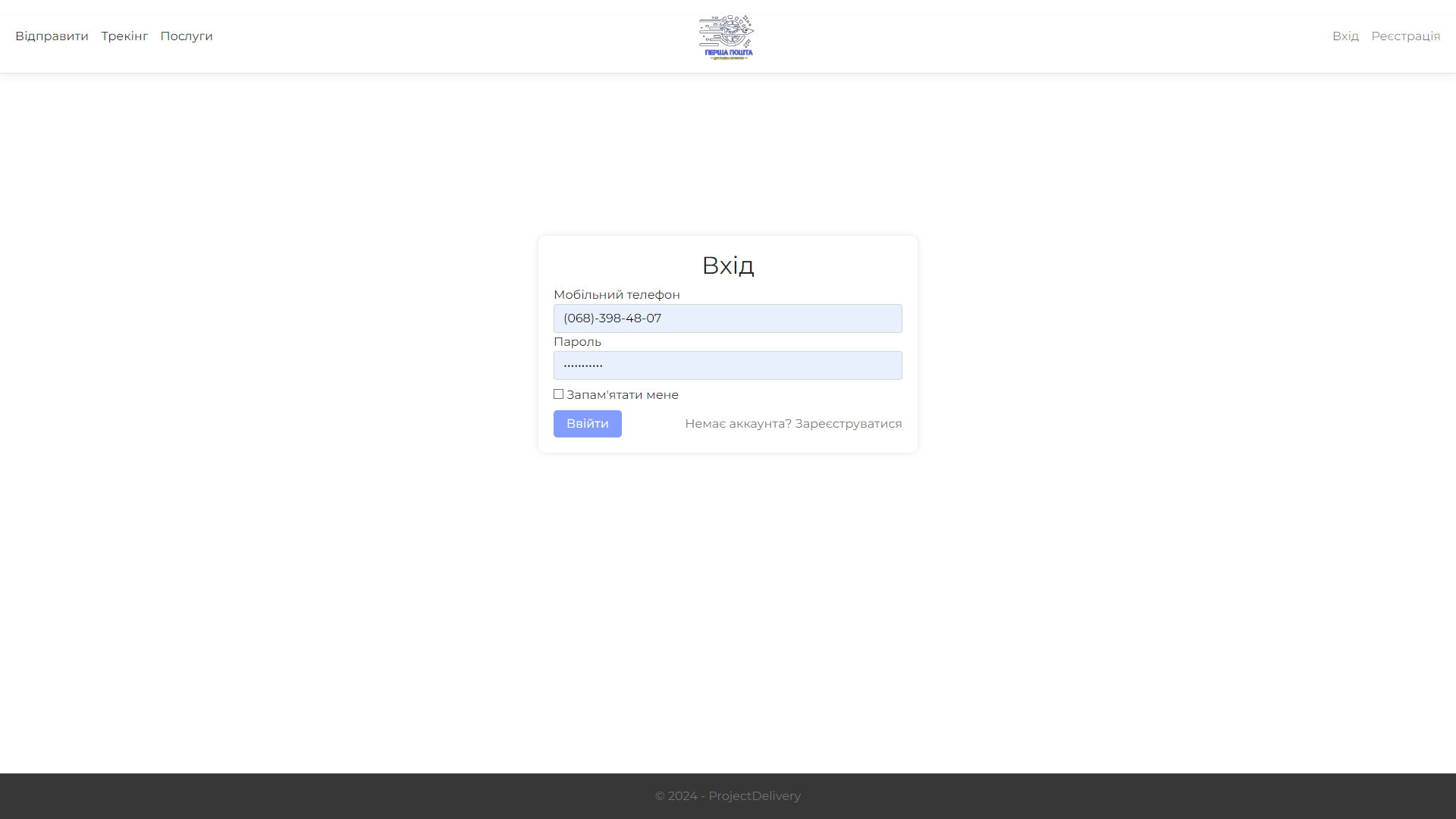


Рисунок 5.9 – Скріншот виконання тесту ЗК.1.1

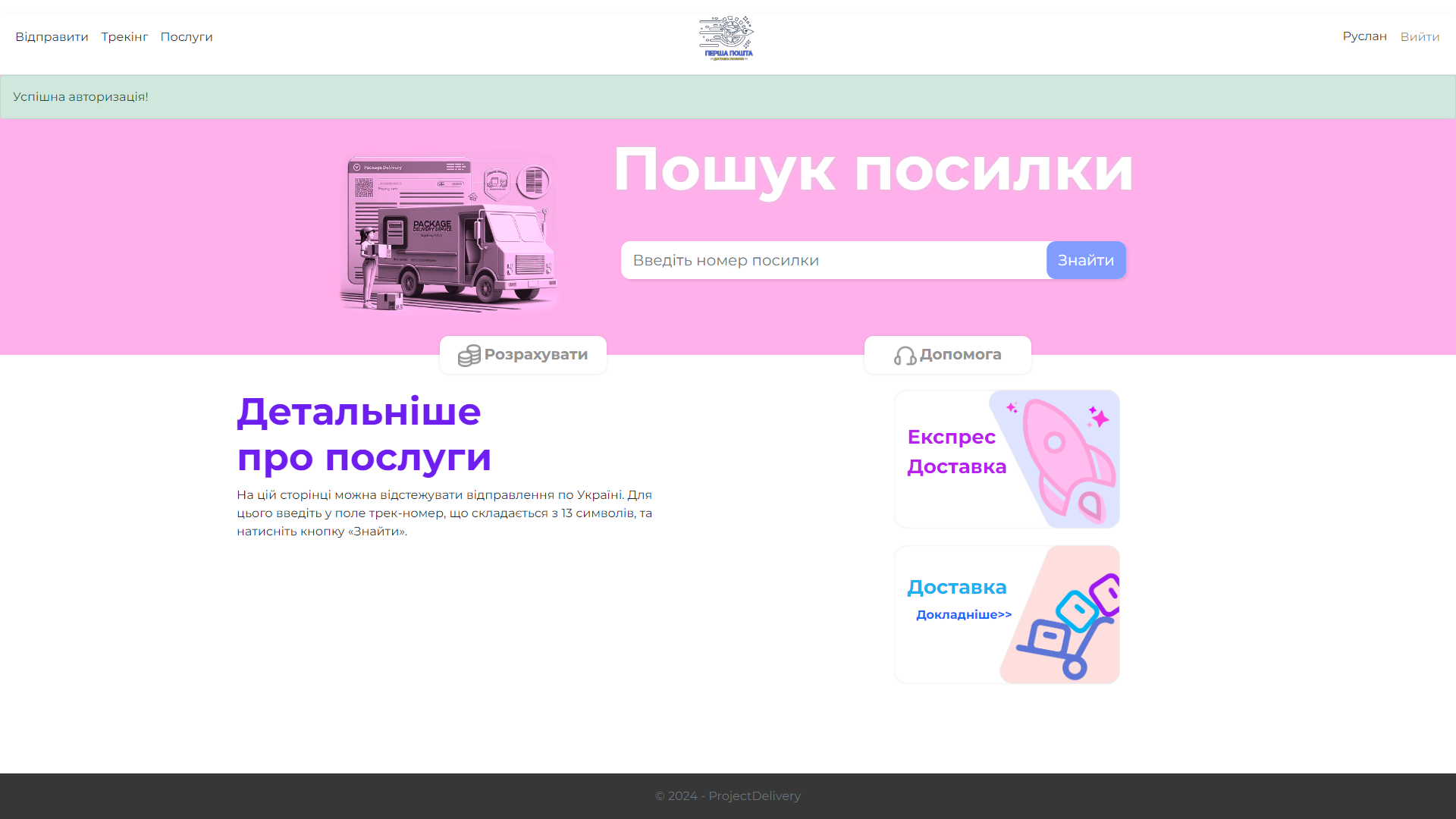


Рисунок 5.10 – Скріншот виконання тесту ЗК.1.1

**Ідентифікатор тесту:** ЗК.1.2.

**Вхідні дані:**

1. Ввести мобільний номер: «0689999999».
2. Ввести неправильний пароль.
3. Натиснути кнопку : «Ввійти».

**Очікуваний результат:** Помилка авторизації

**Отриманий результат:**



Рисунок 5.11 – Скріншот виконання тесту ЗК.1.2

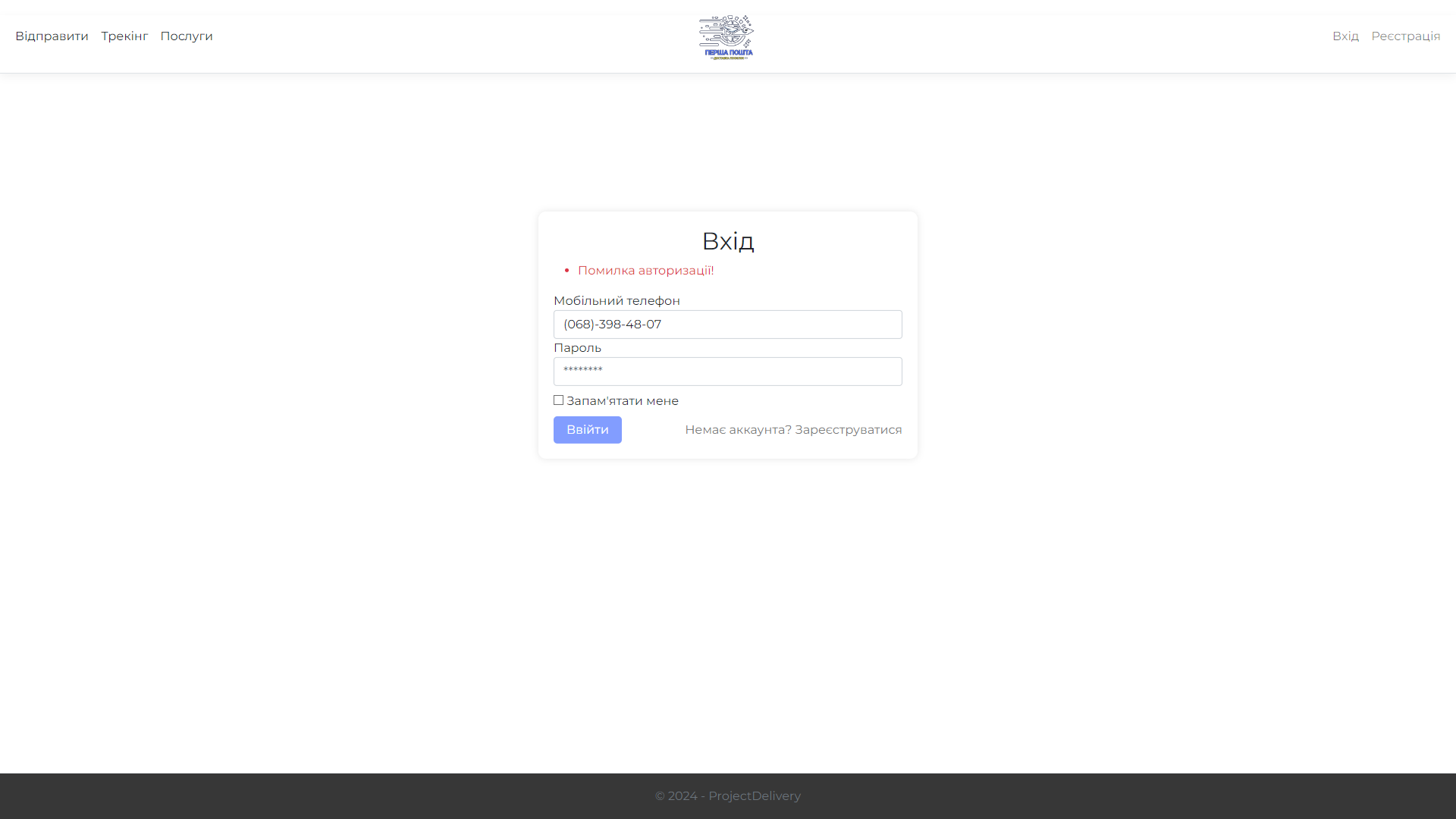


Рисунок 5.12 – Скріншот виконання тесту ЗК.1.2

**Ідентифікатор тесту:** ЗК.2.1.

**Вхідні дані:**

1. Заповнити дані посилки
2. Вибрати доставку (звичайна або швидка).
3. Натиснути кнопку : «Створити посилку».

**Очікуваний результат:** Успішне створення посилки

**Отриманий результат:**

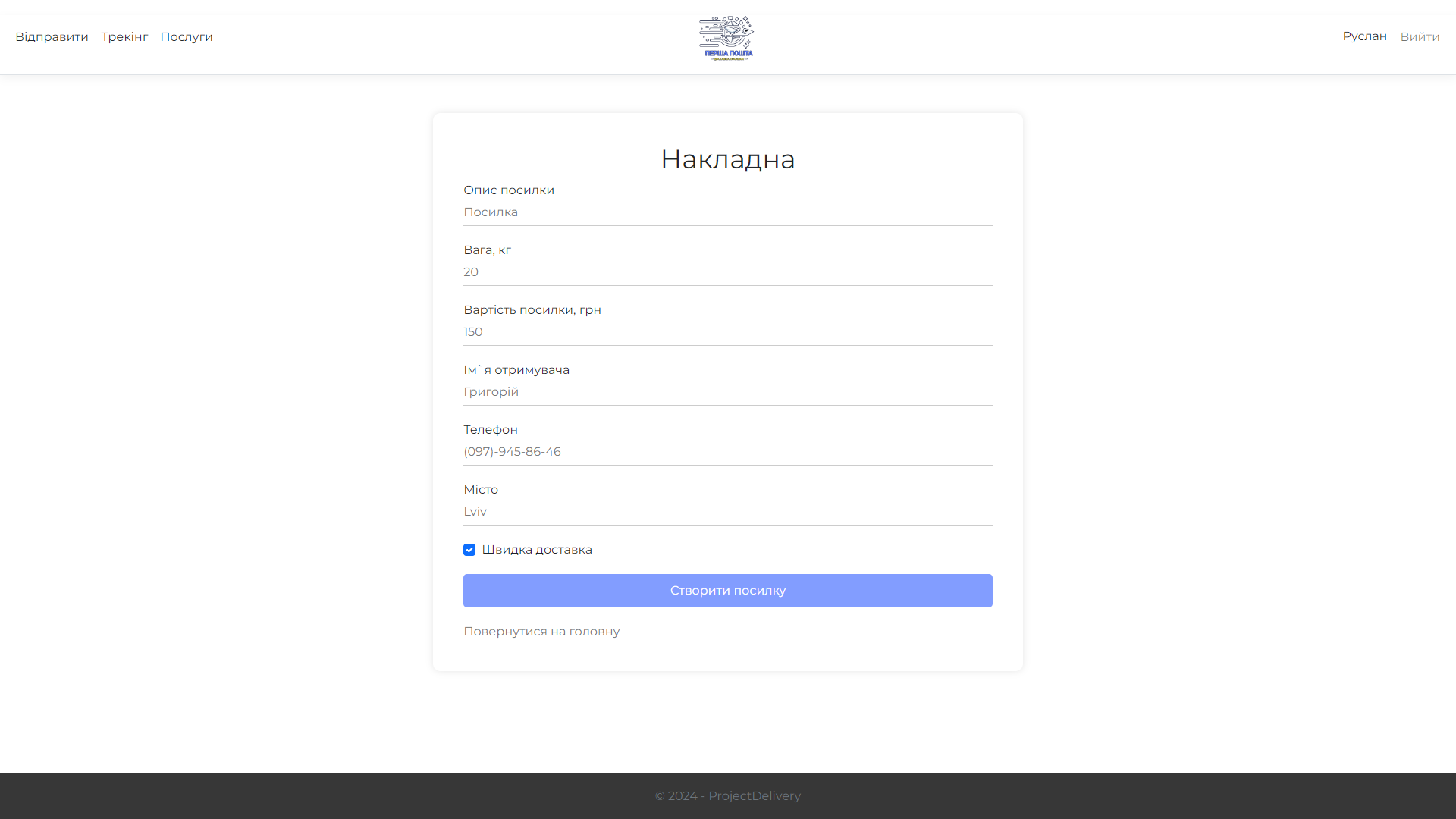


Рисунок 5.13 – Скріншот виконання тесту ЗК.2.1

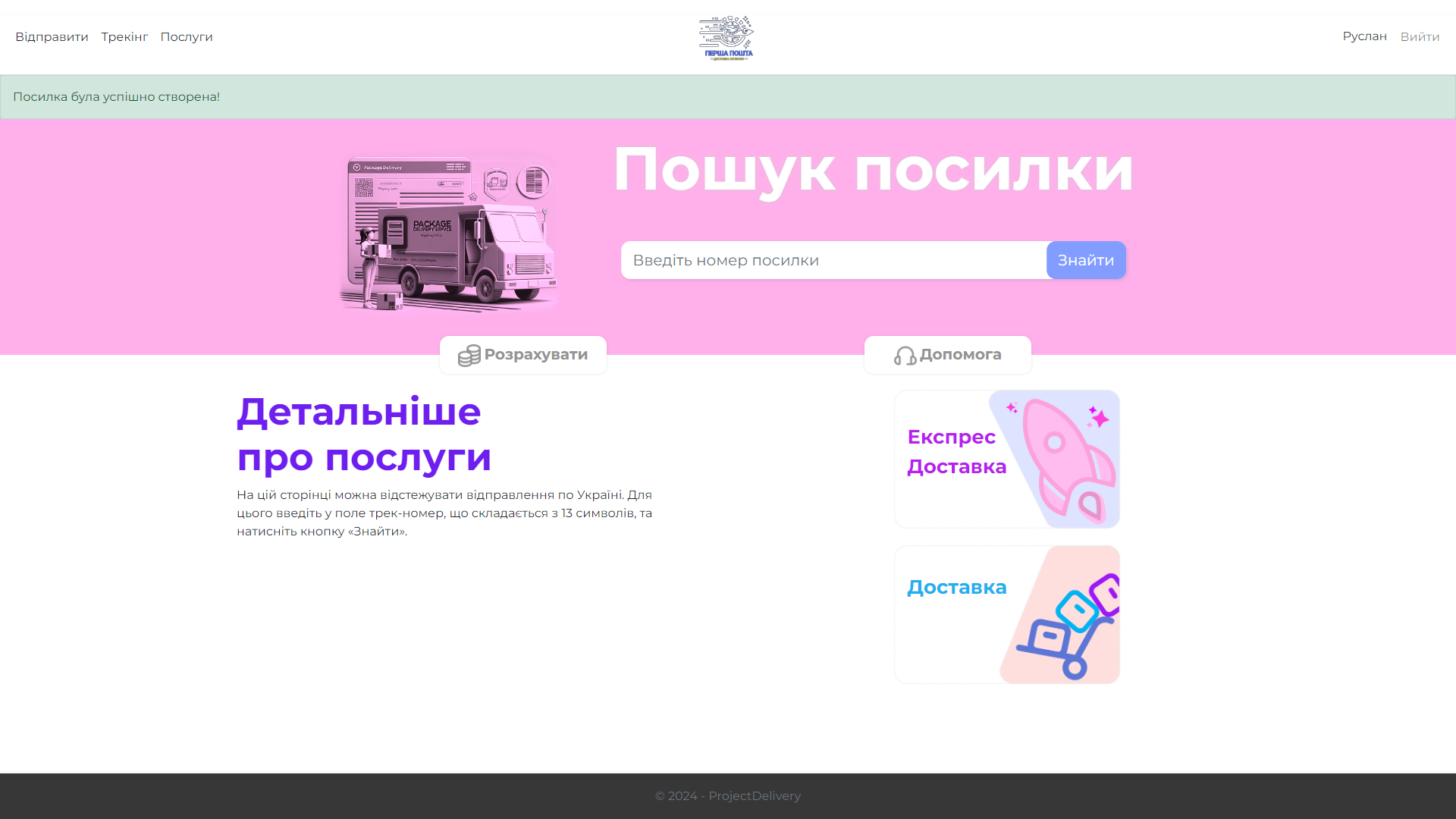


Рисунок 5.14 – Скріншот виконання тесту ЗК.2.1

**Ідентифікатор тесту:** ЗК.3.1.

**Вхідні дані:**

1. Натиснути на особистий профіль.
2. Натиснути на кнопку «Посилки».

**Очікуваний результат:** Виведення інформації щодо посилок

**Отриманий результат:**

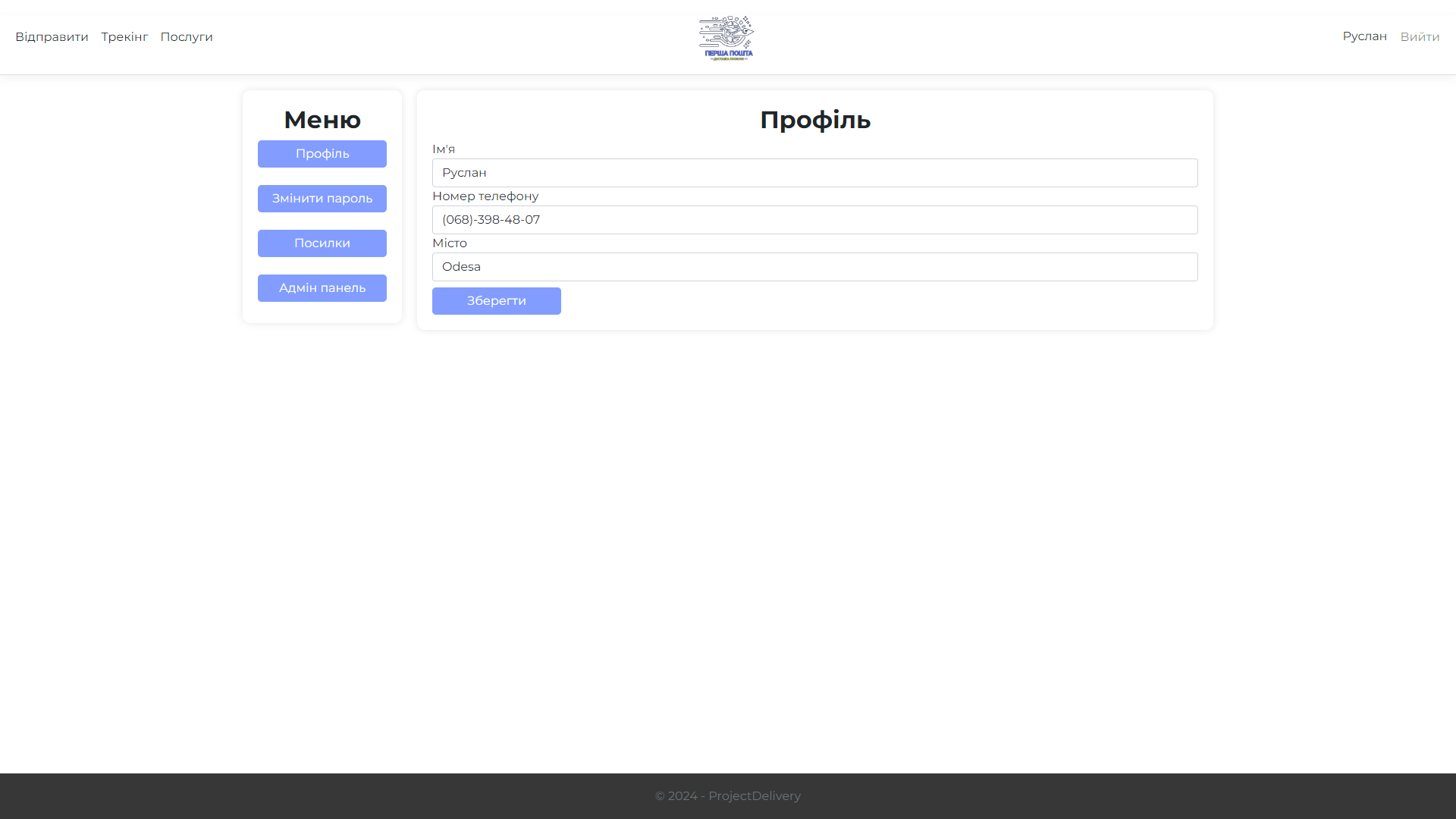


Рисунок 5.15 – Скріншот виконання тесту ЗК.3.1

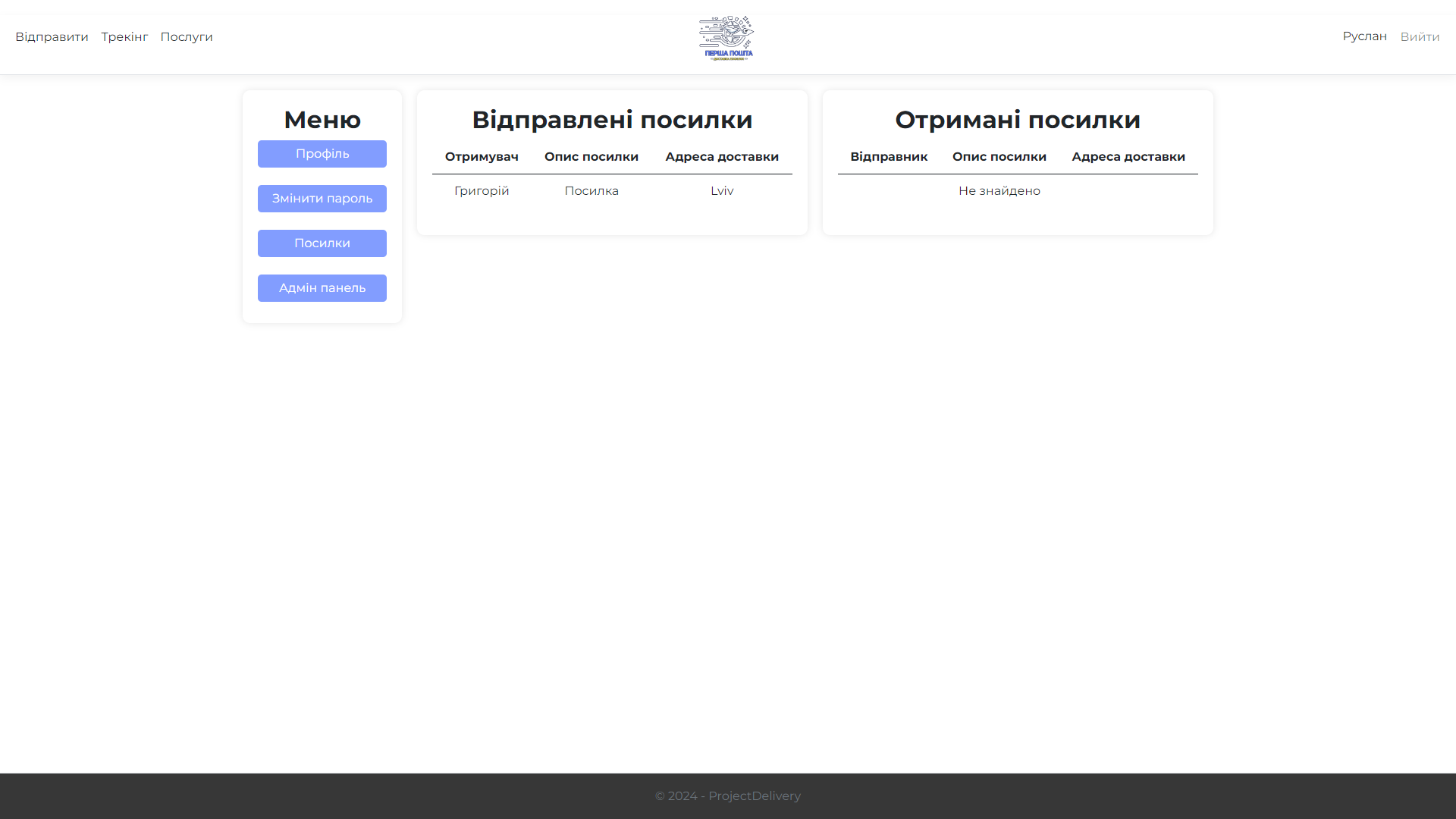


Рисунок 5.16 – Скріншот виконання тесту ЗК.3.1

**Ідентифікатор тесту:** ЗК.4.1.

**Вхідні дані:**

1. Натиснути на особистий профіль.
2. Змінити дані.
3. Натиснути кнопку : «Зберегти».

**Очікуваний результат:** Змінення інформації користувача

**Отриманий результат:**

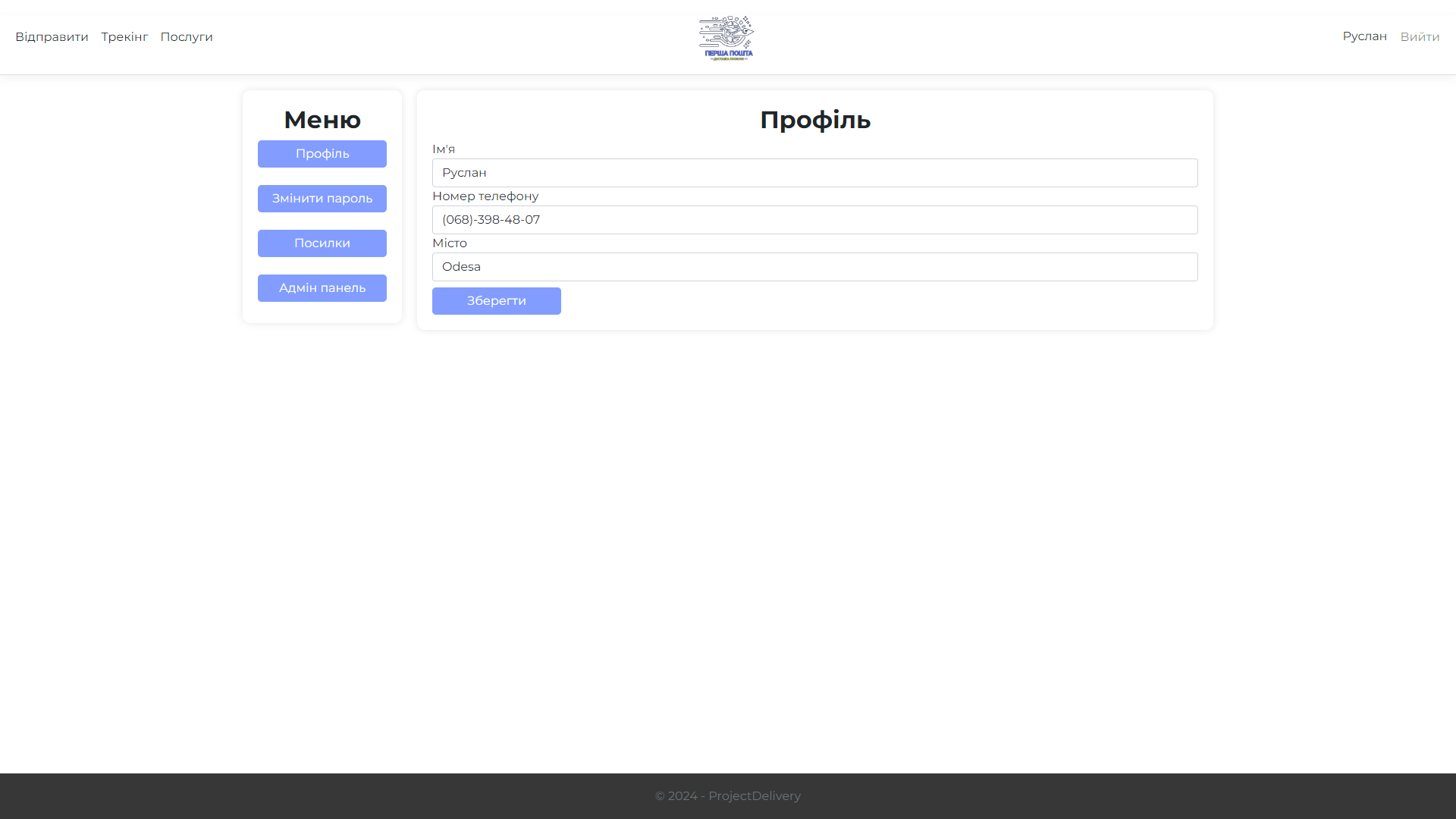


Рисунок 5.17 – Скріншот виконання тесту ЗК.4.1

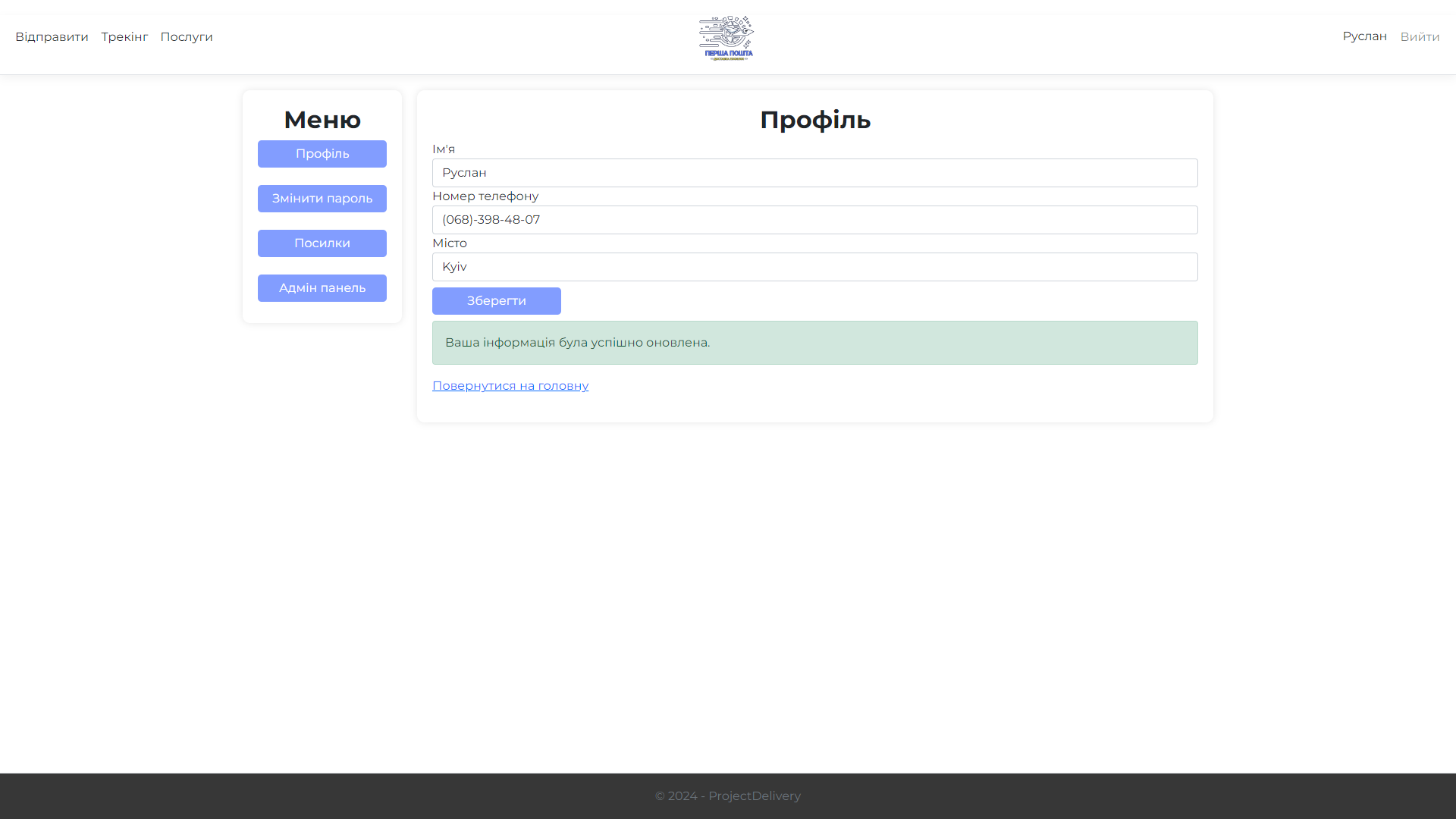


Рисунок 5.18 – Скріншот виконання тесту ЗК.4.1

## Функціональне тестування для ролі користувача «Гість»

Для проведення функціонального тестування для ролі «Гість» були використані розроблені у п. 1.4.3. функціональні тести. Результати проведення функціонального тестування наведені нижче.

**Ідентифікатор тесту:** Г.1.1

**Вхідні дані:**

1. Натиснути на кнопку «Розрахувати».
2. Натиснути на кнопку «Посилки».
3. Ввести дані.
4. Натиснути на кнопку «Розрахувати вартість»

**Очікуваний результат:** Виведення приблизної вартості доставки посилки

**Отриманий результат:**

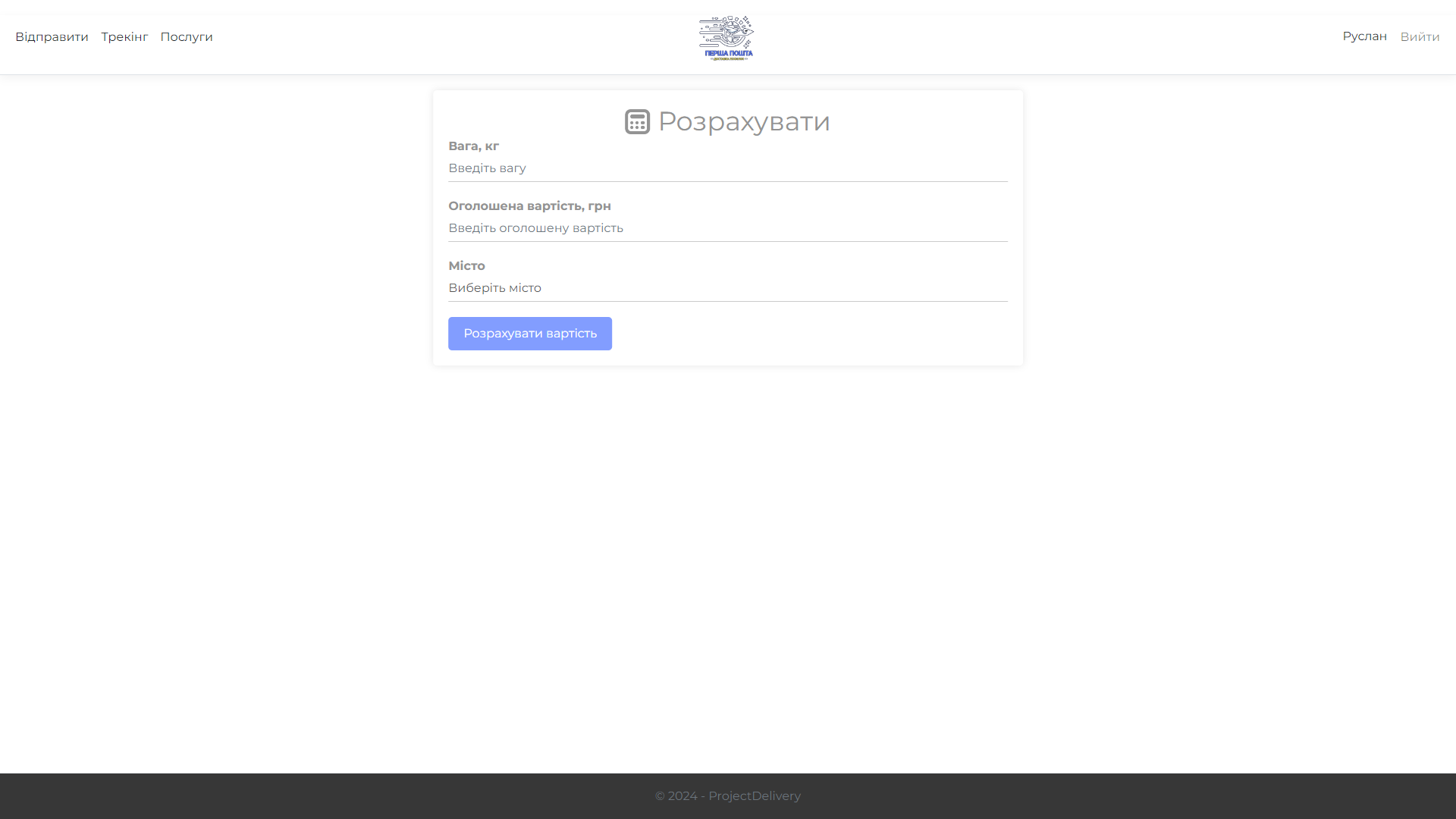


Рисунок 5.19 – Скріншот виконання тесту Г.1.1

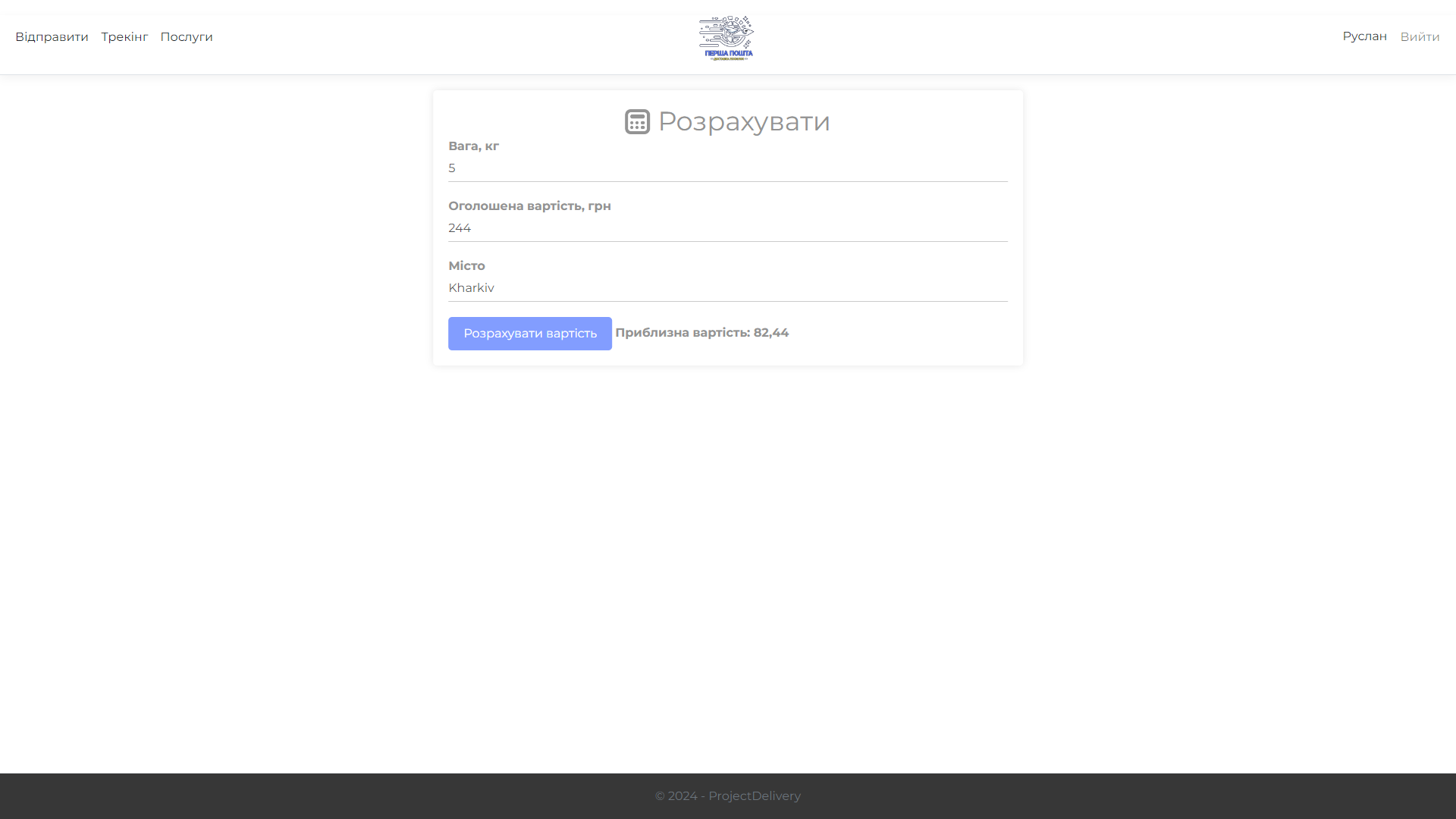


Рисунок 5.20 – Скріншот виконання тесту Г.1.1

**Ідентифікатор тесту:** Г.2.1

**Вхідні дані:**

1. Натиснути на кнопку «Трекінг».
2. Ввести номер існуючої посилки.
3. Натиснути кнопку «Відстежити».

**Очікуваний результат:** Виведення інформації про знайдену посилку

**Отриманий результат:**

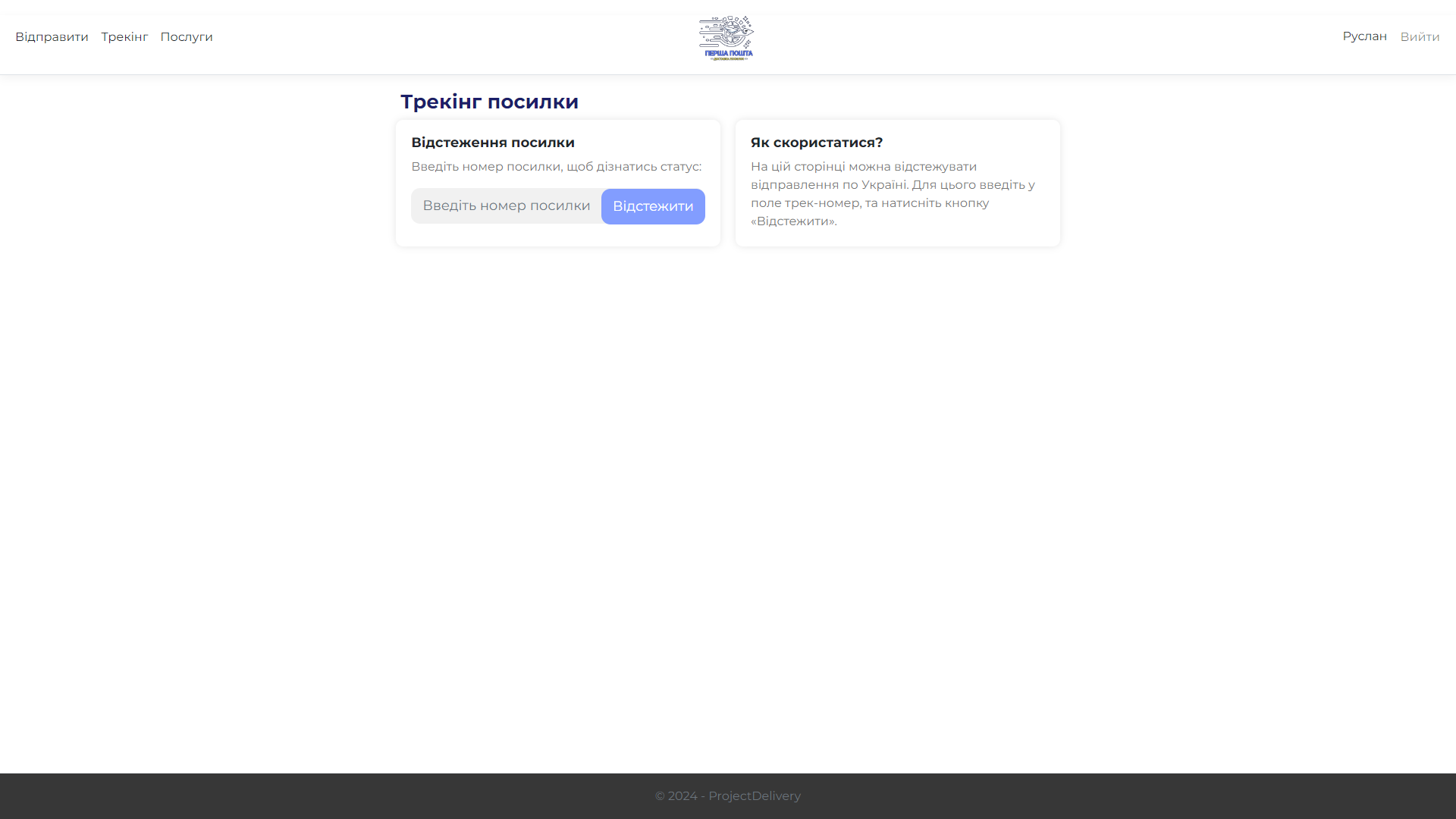


Рисунок 5.21 – Скріншот виконання тесту Г.2.1

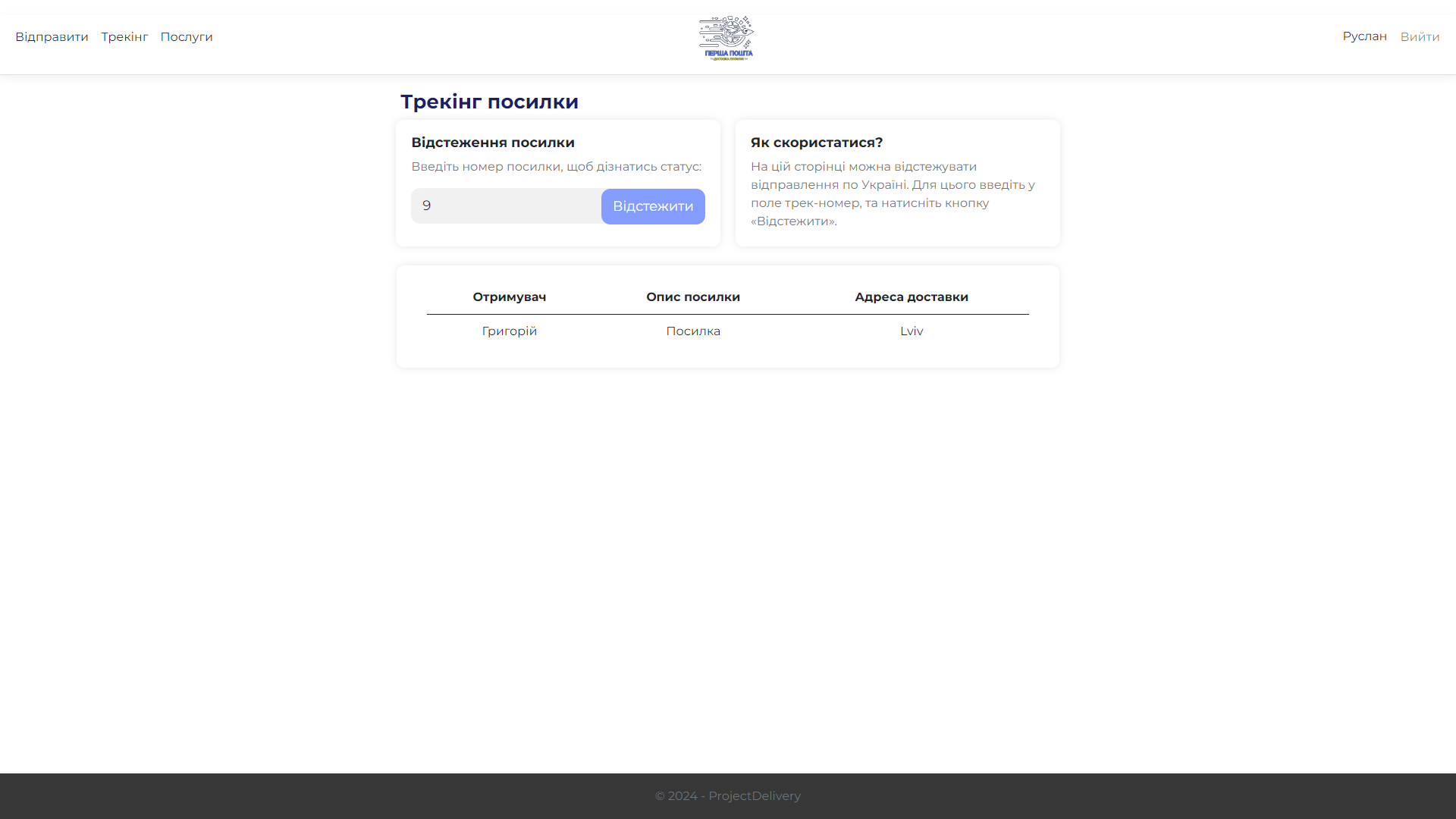


Рисунок 5.22 – Скріншот виконання тесту Г.2.1

**Ідентифікатор тесту:** Г.2.2

**Вхідні дані:**

1. Натиснути на кнопку «Трекінг».
2. Ввести номер неіснуючої посилки.
3. Натиснути кнопку «Відстежити».

**Очікуваний результат:** Виведення повідомлення про те, що посилка не була знайдена

**Отриманий результат:**

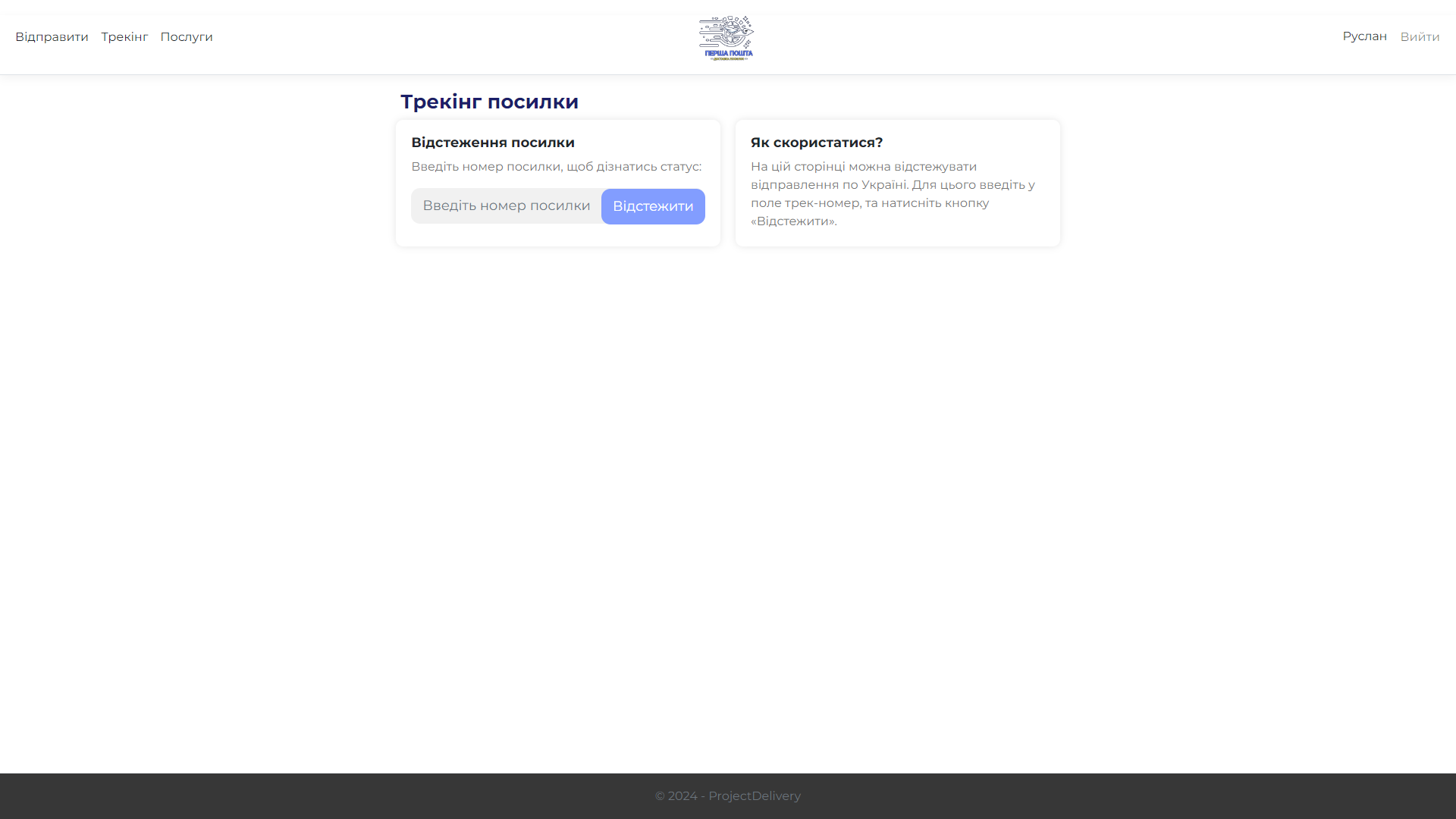


Рисунок 5.23 – Скріншот виконання тесту Г.2.2

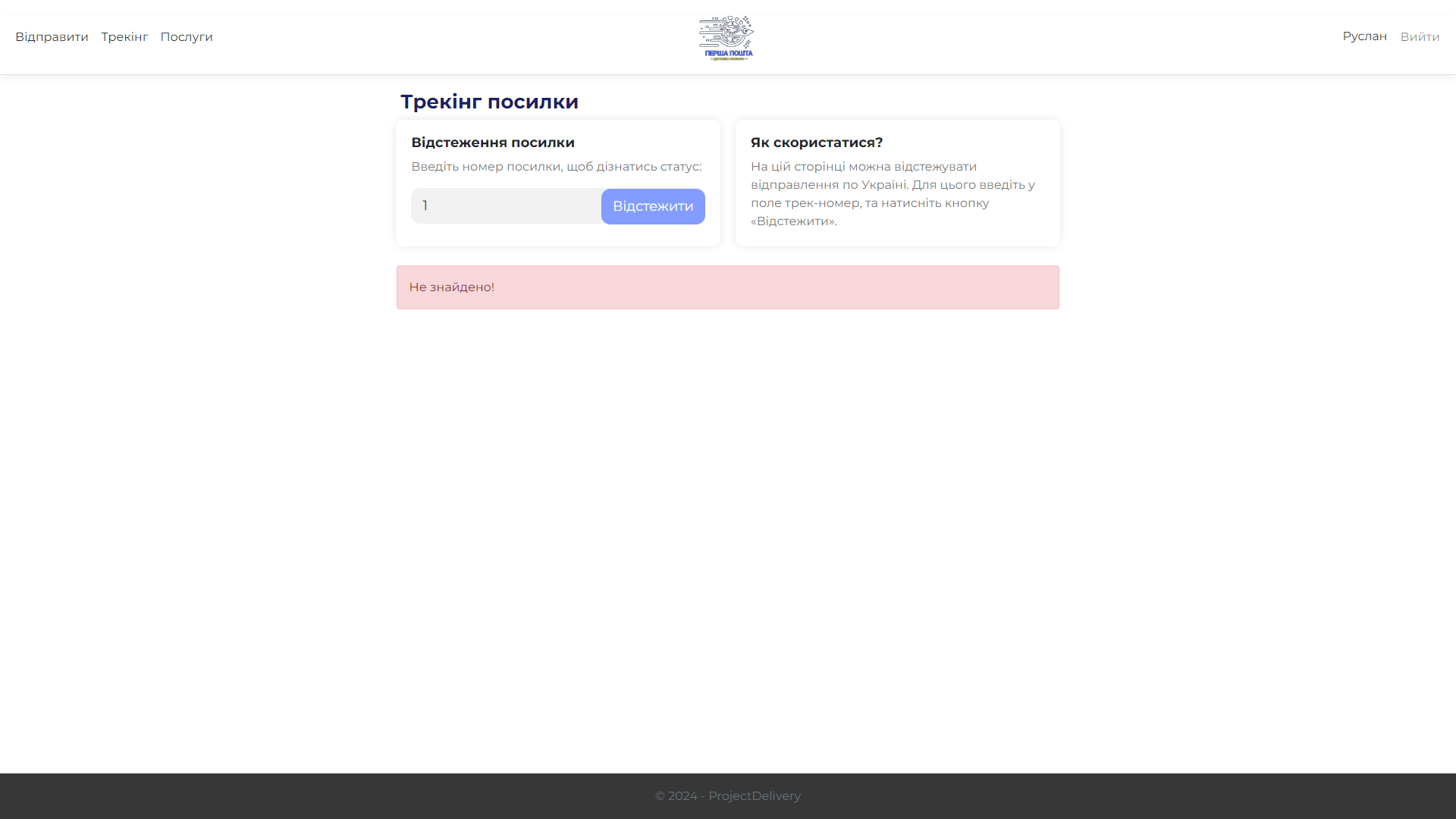


Рисунок 5.24 – Скріншот виконання тесту Г.2.2

**Ідентифікатор тесту:** Г.3.1

**Вхідні дані:**

1. Натиснути на кнопку: «Реєстрація».
2. Ввести ім’я користувача: «Олексій».
3. Ввести пошту: «oleksiy@gmail.com».
4. Ввести мобільний номер: «0979999999».
5. Ввести пароль: «qwerty1234».
6. Підтвердити пароль.

**Очікуваний результат:** Успішна реєстрація користувача

**Отриманий результат:**

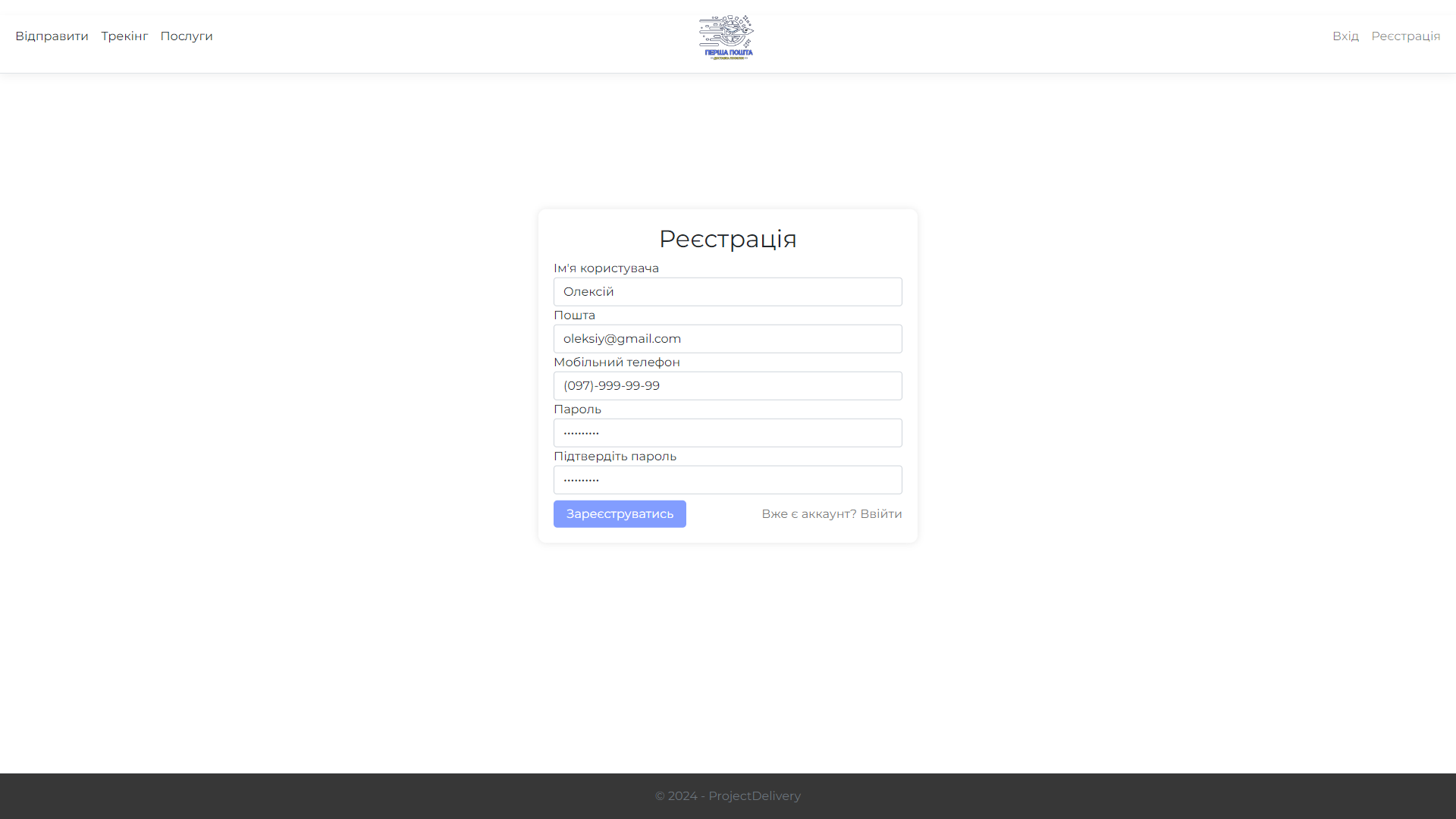


Рисунок 5.25 – Скріншот виконання тесту Г.3.1

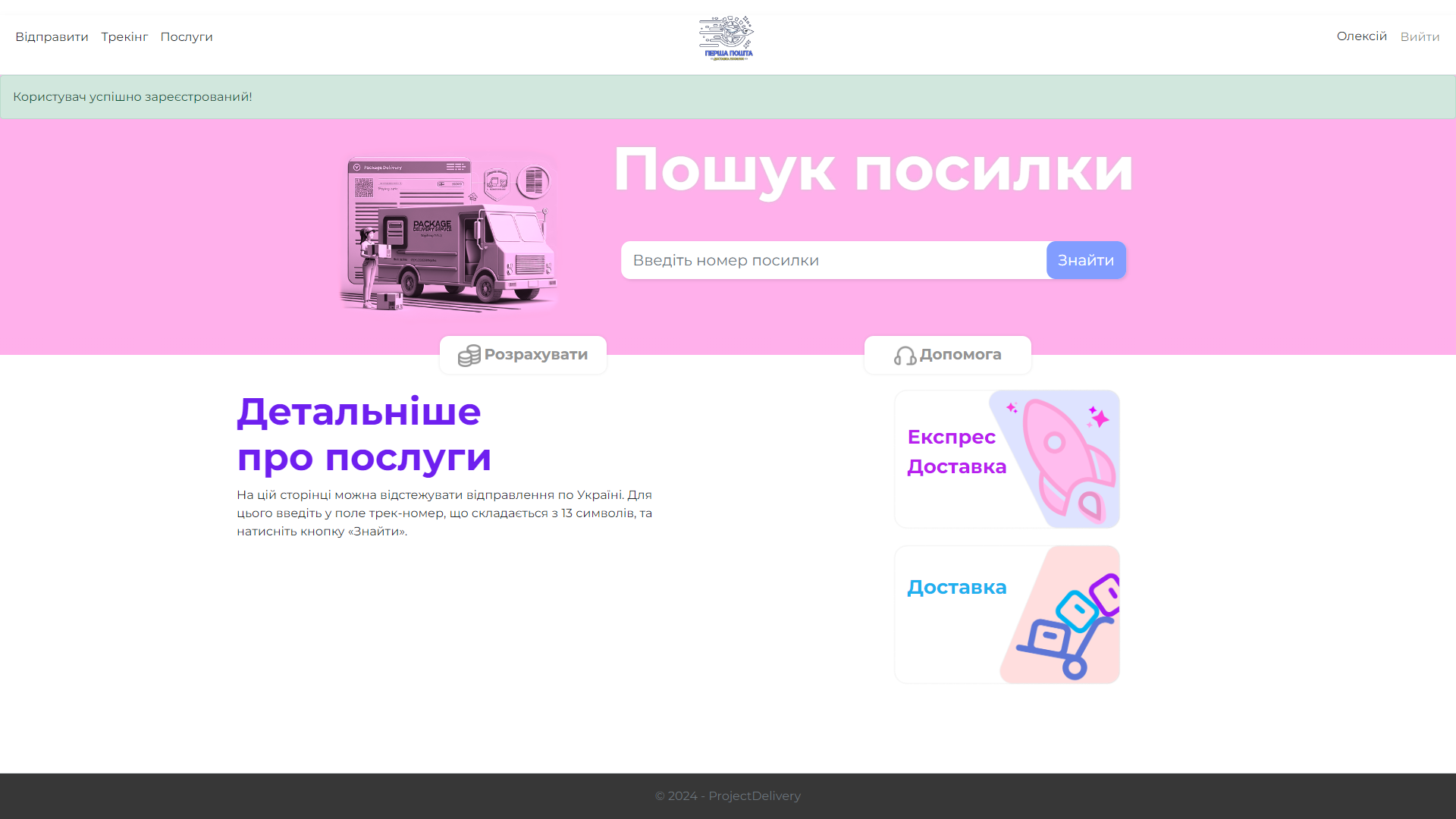


Рисунок 5.26 – Скріншот виконання тесту Г.3.1

**Ідентифікатор тесту:** Г.3.2

**Вхідні дані:**

1. Натиснути на кнопку: «Реєстрація».
2. Ввести ім’я користувача: «Олексій».
3. Ввести пошту: «oleksiy@gmail.com».
4. Ввести мобільний номер: «0979999999».
5. Ввести пароль: «пароль1234».
6. Підтвердити пароль.

**Очікуваний результат:** Користувач не був зареєстрований і була виведена помилка

**Отриманий результат:**

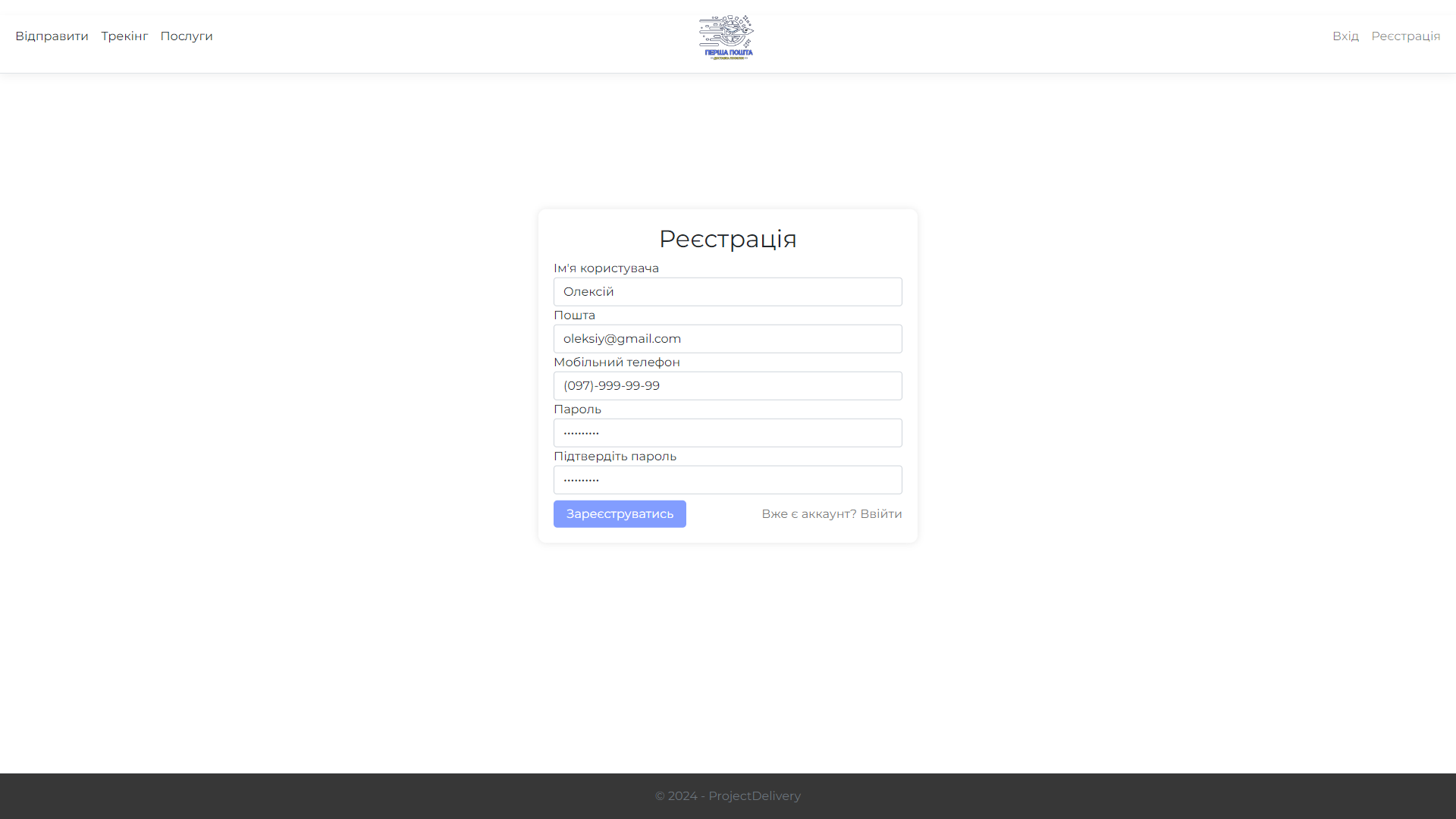


Рисунок 5.27 – Скріншот виконання тесту Г.3.2

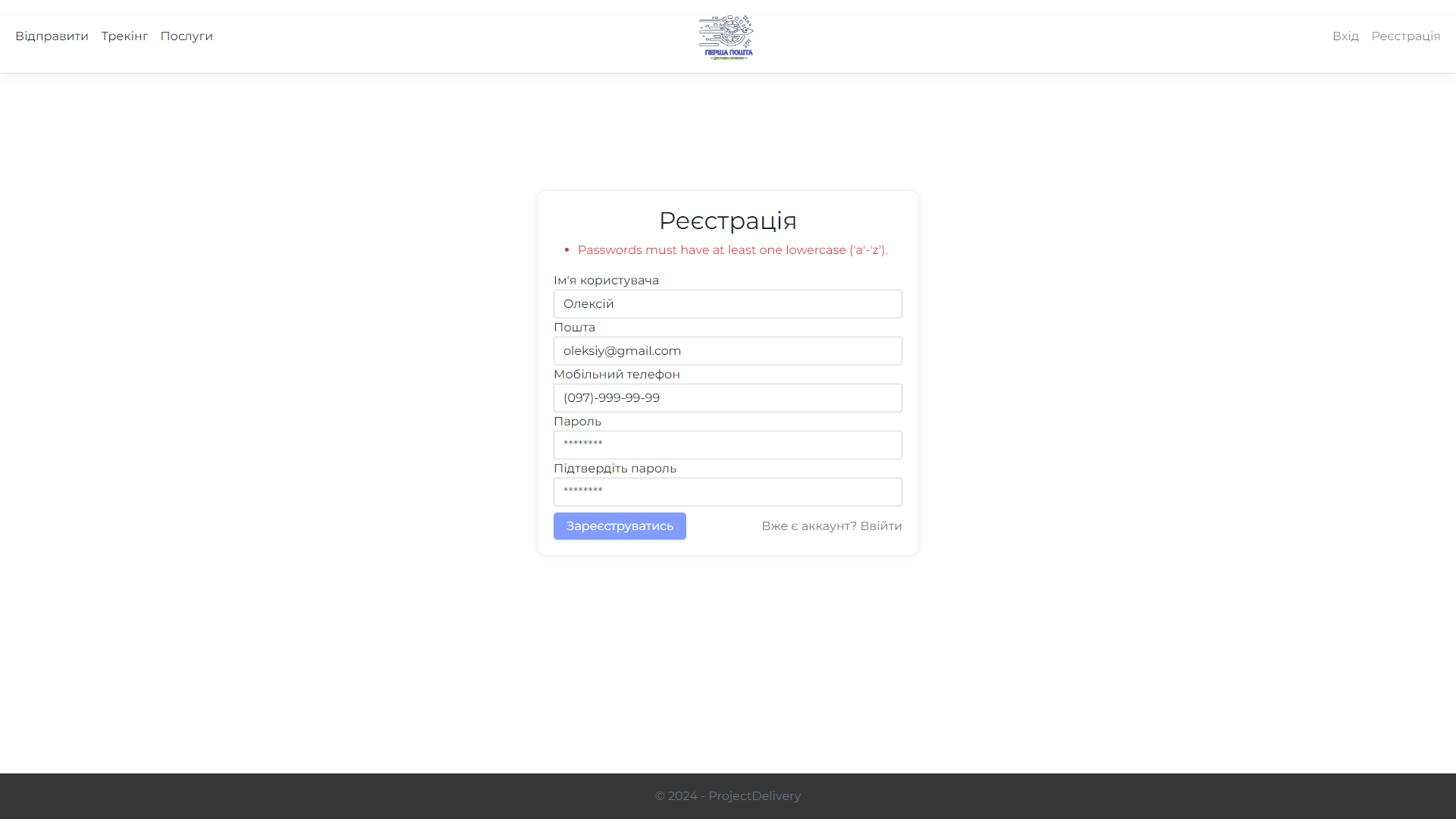


Рисунок 5.28 – Скріншот виконання тесту Г.3.2

# ВИСНОВКИ

У результаті виконання курсової роботи було розроблено програмне забезпечення з використанням об’єктно-орієнтованої парадигми для «Логістичні послуги».

В ході роботи було створено web-застосунок використовуючи ASP .NET Core MVC (модель-представлення-контролер) та Entity Framework для LINQ запитів до БД SQL Server. Було реалізовано реєстрацію, авторизацію користувача, створення та відстеження посилок. Для користувача з роллю «Admin», було створено панель адміністратора для керування користувачами та статями.

Оформлено звіт.

**Примітка:**

Посилання на github-репозиторій: <https://github.com/rusgord/Course-Work.git>

Посилання на відеоролик: <https://drive.google.com/file/d/17nVjyPN-iMChR6oe8DjRKL1YaAC0ot9x/view?usp=drive_link>

# ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. Воробйов, Ю. А. Правила оформлення навчальних і науково-дослідних документів [Текст] : навч. посіб. / Ю. А. Воробйов, Ю. О. Сисоєв. – 4-те вид., випр. і доп. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2019. – 88 с.
2. Автоматичне оформлення джерел по ВАК України [Електронний ресурс] – Режим доступу до ресурсу: <http://vak.in.ua/do.php>.
3. Нова Пошта [Електронний ресурс] – Режим доступу до ресурсу: <https://novapost.com/uk-ua/>.
4. Укрпошта [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ukrposhta.ua/ua>.

# ДОДАТОК А. Лістинг класів предметної області

Клас «AccModel»:

using Microsoft.AspNetCore.Identity;

using ProjectDelivery.Enums;

namespace ProjectDelivery.Models

{

public class AccModel : IdentityUser

{

public string Name { get; set; }

public EnumRoles Roles { get; set; } = 0;

public EnumCities City { get; set; } = 0;

}

}

Клас «AddUserModel»:

using ProjectDelivery.Enums;

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class AddUserModel

{

[Required]

[Display(Name = "Name")]

public string Name { get; set; }

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[Phone]

[Display(Name = "Phone")]

public string PhoneNumber { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]

[Display(Name = "Password")]

public string Password { get; set; }

public EnumRoles Roles { get; set; } = 0;

public EnumCities City { get; set; } = 0;

}

}

Клас «AdminPanelViewModel»:

namespace ProjectDelivery.Models

{

public class AdminPanelViewModel

{

public List<AccModel>? accModels { get; set; }

public List<ArticleModel>? articleModels { get; set; }

}

}

Клас «ArticleModel»:

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class ArticleModel

{

public int Id { get; set; }

[Required]

[StringLength(50, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 3)]

public string Title { get; set; }

[StringLength(3000, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 5)]

public string Description { get; set; }

}

}

Клас «CalcModel»:

using ProjectDelivery.Enums;

namespace ProjectDelivery.Models

{

public class CalcModel

{

public double Weight { get; set; }

public double Cost { get; set; }

public EnumCities City { get; set; }

public double Price { get; set; } = 0;

}

}

Клас «ChangePasswordViewModel»:

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class ChangePasswordViewModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Ваш пароль")]

public string CurrentPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Новий пароль")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Підтвердіть новий пароль")]

[Compare("NewPassword", ErrorMessage = "Паролі не співпадають!")]

public string ConfirmPassword { get; set; }

public bool Changed { get; set; } = false;

}

}

Клас «ErrorViewModel»:

namespace ProjectDelivery.Models

{

public class ErrorViewModel

{

public string? RequestId { get; set; }

public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

}

}

Клас «LoginViewModel»:

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class LoginViewModel

{

[Required]

[Phone]

[Display(Name = "Phone")]

public string Phone { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me")]

public bool RememberMe { get; set; }

}

}

Клас «Package»:

using ProjectDelivery.Enums;

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class Package

{

public int Id { get; set; }

public string? Description { get; set; }

public bool TypePayment { get; set; }

[Required]

public int Weight { get; set; }

[Required]

public int Price { get; set; }

public string? recipientName { get; set; }

public string? senderName { get; set; }

[Required]

public string Phone { get; set; }

public EnumCities City { get; set; }

public string? SenderId { get; set; } = null;

public string? RecipientId { get; set; } = null;

}

}

Клас «RegisterViewModel»:

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class RegisterViewModel

{

[Required]

[Display(Name = "Name")]

public string Name { get; set; }

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[Phone]

[Display(Name = "Phone")]

public string PhoneNumber { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

}

Клас «TrackingModel»:

using ProjectDelivery.Enums;

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class TrackingModel

{

[Required]

public int Id { get; set; }

public bool IsFind { get; set; }

public string? Text { get; set; }

public string? Name { get; set; }

public string? Description { get; set; }

public string? City { get; set; }

}

}

Клас «ViewProfileViewModel»:

using ProjectDelivery.Enums;

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class ViewProfileViewModel

{

[Required]

[Display(Name = "Ім'я")]

public string Name { get; set; }

[Required]

[Phone]

[Display(Name = "Номер телефону")]

public string PhoneNumber { get; set; }

public EnumCities City { get; set; } = 0;

public bool Changed { get; set; } = false;

}

}

# ДОДАТОК Б. Лістинг класів і інтерфейсів

Інтерфейс IAccount:

public interface IAccount

{

public IActionResult Register();

public IActionResult Login(string returnUrl);

public Task<IActionResult> Logout();

public IActionResult ChangePassword();

public Task<IActionResult> ViewProfile();

public Task<IActionResult> HistoryPackages();

}

Інтерфейс IAdmin:

public interface IAdmin

{

public Task<IActionResult> AdminPanel();

public Task<IActionResult> AddUser();

public Task<IActionResult> UpdateUser(string userId, string Name, string Email, string Phone, EnumRoles Role);

public Task<IActionResult> DeleteUser(string userId);

public Task<IActionResult> AddArticle();

public Task<IActionResult> UpdateArticle(int articleId, string Title, string Description);

public Task<IActionResult> DeleteArticle(int articleId);

}

Класс AccModel:

using Microsoft.AspNetCore.Identity;

using ProjectDelivery.Enums;

namespace ProjectDelivery.Models

{

public class AccModel : IdentityUser

{

public string Name { get; set; }

public EnumRoles Roles { get; set; } = 0;

public EnumCities City { get; set; } = 0;

}

}

Класс Package:

using ProjectDelivery.Enums;

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class Package

{

public int Id { get; set; }

public string? Description { get; set; }

public bool TypePayment { get; set; }

[Required]

public int Weight { get; set; }

[Required]

public int Price { get; set; }

public string? recipientName { get; set; }

public string? senderName { get; set; }

[Required]

[Phone]

public string Phone { get; set; }

public EnumCities City { get; set; }

public string? SenderId { get; set; } = null;

public string? RecipientId { get; set; } = null;

}

}

Клас «ArticleModel»:

using System.ComponentModel.DataAnnotations;

namespace ProjectDelivery.Models

{

public class ArticleModel

{

public int Id { get; set; }

[Required]

[StringLength(50, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 3)]

public string Title { get; set; }

[StringLength(3000, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 5)]

public string Description { get; set; }

}

}

# ДОДАТОК В. Лістинг класів unit-тестів

Клас unit-тесту «PackageTest»

using ProjectDelivery.Models;

using ProjectDelivery.Enums;

namespace Project\_Test

{

public class PackageTest

{

[Fact]

public void PackageModel()

{

var package = new Package

{

Id = 1,

TypePayment = true,

Weight = 10,

Price = 10,

Phone = "0998953467",

City = EnumCities.Dnipro

};

Assert.Equal(1, package.Id);

Assert.Equal(true, package.TypePayment);

Assert.Equal(10, package.Weight);

Assert.Equal(10, package.Price);

Assert.Equal("0998953467", package.Phone);

Assert.Equal(EnumCities.Dnipro, package.City);

}

[Fact]

public void PackageTrackingModel()

{

var track\_package = new TrackingModel

{

Id = 1,

IsFind = true

};

Assert.Equal(1, track\_package.Id);

Assert.Equal(true, track\_package.IsFind);

}

}

}

Клас unit-тесту «AccountTest»

using ProjectDelivery.Models;

using ProjectDelivery.Enums;

using Microsoft.AspNetCore.Identity;

namespace Project\_Test

{

public class AccountTest

{

[Fact]

public void AccModel()

{

var acc = new AccModel

{

Name = "Test",

Roles = EnumRoles.User,

City = EnumCities.Kharkiv,

PhoneNumber = "0999644686",

Email = "user@mail.com",

PasswordHash = "aljkf834jfdk;fo8w"

};

Assert.Equal("Test", acc.Name);

Assert.Equal(EnumRoles.User, acc.Roles);

Assert.Equal(EnumCities.Kharkiv, acc.City);

Assert.Equal("0999644686", acc.PhoneNumber);

Assert.Equal("user@mail.com", acc.Email);

Assert.Equal("aljkf834jfdk;fo8w", acc.PasswordHash);

}

[Fact]

public void AccLoginModel()

{

var acc = new AccModel

{

Name = "Test",

Roles = EnumRoles.User,

City = EnumCities.Kharkiv,

PhoneNumber = "0999644686",

Email = "user@mail.com",

PasswordHash = "aljkf834jfdk;fo8w"

};

var login = new LoginViewModel{

Phone = acc.PhoneNumber,

Password = acc.PasswordHash,

RememberMe = false

};

Assert.Equal(acc.PhoneNumber, login.Phone);

Assert.Equal(acc.PasswordHash, login.Password);

Assert.Equal(false, login.RememberMe);

}

[Fact]

public void AccRegisterModel()

{

var acc = new AccModel

{

Name = "Test",

Roles = EnumRoles.User,

City = EnumCities.Kharkiv,

PhoneNumber = "0999644686",

Email = "user@mail.com",

PasswordHash = "aljkf834jfdk;fo8w"

};

var register = new RegisterViewModel

{

Name = acc.Name,

Email = acc.Email,

PhoneNumber= acc.PhoneNumber,

Password = acc.PasswordHash,

ConfirmPassword = "aljkf834jfdk;fo8w"

};

Assert.Equal(acc.Name, register.Name);

Assert.Equal(acc.Email, register.Email);

Assert.Equal(acc.PhoneNumber, register.PhoneNumber);

Assert.Equal(acc.PasswordHash, register.Password);

Assert.Equal("aljkf834jfdk;fo8w", register.ConfirmPassword);

}

}

}

Клас unit-тесту «ArtcleTest»

using ProjectDelivery.Models;

using Microsoft.AspNetCore.Identity;

using ProjectDelivery.Enums;

namespace Project\_Test

{

public class ArticleTest

{

[Fact]

public void ArticleModel()

{

var article = new ArticleModel

{

Id = 1,

Title = "Test title",

Description = "Some text"

};

Assert.Equal(1, article.Id);

Assert.Equal("Test title", article.Title);

Assert.Equal("Some text", article.Description);

}

}

}

# ДОДАТОК Г. Лістинг класів інтерфейсу користувача

Клас «AccountContlroller»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ProjectDelivery.Classes;

using ProjectDelivery.Data;

using ProjectDelivery.Interfaces;

using ProjectDelivery.Models;

namespace ProjectDelivery.Controllers

{

public class AccountController : Controller, IAccount

{

private readonly UserManager<AccModel> \_userManager;

private readonly SignInManager<AccModel> \_signInManager;

private readonly ApplicationDbContext \_context;

public delegate Task UserEventHandler(string userId, string message);

public event UserEventHandler UserEventOccurred;

public AccountController(UserManager<AccModel> userManager, SignInManager<AccModel> signInManager, ApplicationDbContext context)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_context = context;

UserEventOccurred += async (userId, message) =>

{

TempData["UserNotification"] = message;

};

}

[HttpGet]

[AllowAnonymous]

public IActionResult Register()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var existingUser = await new FindingUser().FindByPhoneNumberAsync(\_userManager, model.PhoneNumber);

if (existingUser != null)

{

ModelState.AddModelError(string.Empty, "Цей телефон вже використовується!");

return View(model);

}

var user = new AccModel { Name = model.Name, UserName = model.Email, Email = model.Email, PhoneNumber = model.PhoneNumber };

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await \_signInManager.SignInAsync(user, isPersistent: false);

await UserEventOccurred?.Invoke(user.Id, "Користувач успішно зареєстрований!");

return RedirectToAction("Index", "Home");

}

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

return View(model);

}

[HttpGet]

[AllowAnonymous]

public IActionResult Login(string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

var user = await new FindingUser().FindByPhoneNumberAsync(\_userManager, model.Phone);

if (user != null && await \_userManager.CheckPasswordAsync(user, model.Password))

{

await \_signInManager.SignInAsync(user, model.RememberMe);

await UserEventOccurred?.Invoke(user.Id, "Успішна авторизація!");

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError(string.Empty, "Помилка авторизації!");

return View(model);

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

await \_signInManager.SignOutAsync();

return RedirectToAction("Index", "Home");

}

[HttpGet]

[Authorize]

public IActionResult ChangePassword()

{

ChangePasswordViewModel model = new ChangePasswordViewModel();

return View(model);

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return RedirectToAction(nameof(Login));

}

var changePasswordResult = await \_userManager.ChangePasswordAsync(user, model.CurrentPassword, model.NewPassword);

if (!changePasswordResult.Succeeded)

{

foreach (var error in changePasswordResult.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

return View(model);

}

await \_signInManager.RefreshSignInAsync(user);

model.Changed = true;

return View(model);

}

return View(model);

}

[HttpGet]

[Authorize]

public async Task<IActionResult> ViewProfile()

{

var user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return RedirectToAction(nameof(Login));

}

var model = new ViewProfileViewModel

{

Name = user.Name,

PhoneNumber = user.PhoneNumber,

City = user.City

};

return View(model);

}

[HttpPost]

[Authorize]

[ValidateAntiForgeryToken]

public async Task<IActionResult> ViewProfile(ViewProfileViewModel model)

{

var user = await \_userManager.GetUserAsync(User);

if (ModelState.IsValid)

{

user.Name = model.Name;

user.PhoneNumber = model.PhoneNumber;

user.City = model.City;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

model.Changed = true;

return View(model);

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

[HttpGet]

[Authorize]

public async Task<IActionResult> HistoryPackages()

{

var user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return RedirectToAction(nameof(Login));

}

List<Package> packages = \_context.Packages.ToList();

return View(packages);

}

}

}

Клас «AdminContlroller»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ProjectDelivery.Classes;

using ProjectDelivery.Data;

using ProjectDelivery.Enums;

using ProjectDelivery.Interfaces;

using ProjectDelivery.Models;

namespace ProjectDelivery.Controllers

{

public class AdminController : Controller, IAdmin

{

private readonly UserManager<AccModel> \_userManager;

private readonly ApplicationDbContext \_context;

public AdminController(UserManager<AccModel> userManager, ApplicationDbContext context)

{

\_userManager = userManager;

\_context = context;

}

[HttpGet]

public async Task<IActionResult> AdminPanel()

{

var user = await \_userManager.GetUserAsync(User);

if (user != null && user.Roles == Enums.EnumRoles.Admin)

{

var allUsers = \_userManager.Users.ToList();

var usersExceptCurrent = allUsers.Where(u => u.Id != user.Id).ToList();

var articles = \_context.Articles.ToList();

AdminPanelViewModel model = new() {

accModels = usersExceptCurrent,

articleModels = articles

};

return View(model);

}

return RedirectToAction("Index", "Home");

}

[Authorize]

public async Task<IActionResult> AddUser()

{

AddUserModel newUser = new AddUserModel();

return View(newUser);

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddUser(AddUserModel newUser)

{

if (ModelState.IsValid)

{

var existingUser = await new FindingUser().FindByPhoneNumberAsync(\_userManager, newUser.PhoneNumber); ;

if (existingUser != null)

{

ModelState.AddModelError(string.Empty, "Цей телефон вже використовується!");

return View(newUser);

}

var user = new AccModel { Name = newUser.Name, UserName = newUser.Email, Email = newUser.Email, PhoneNumber = newUser.PhoneNumber, Roles = newUser.Roles, City = newUser.City };

var result = await \_userManager.CreateAsync(user, newUser.Password);

if (result.Succeeded)

{

return RedirectToAction("AdminPanel");

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(newUser);

}

[HttpPost]

[Authorize]

public async Task<IActionResult> UpdateUser(string userId, string Name, string Email, string Phone, EnumRoles Role)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

user.Name = Name;

user.Email = Email;

user.PhoneNumber = Phone;

user.Roles = Role;

await \_userManager.UpdateAsync(user);

await \_context.SaveChangesAsync();

return RedirectToAction("AdminPanel");

}

return RedirectToAction("AdminPanel");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> DeleteUser(string userId)

{

var user = await \_userManager.FindByIdAsync(userId);

if (user != null)

{

await \_userManager.DeleteAsync(user);

await \_context.SaveChangesAsync();

return RedirectToAction("AdminPanel");

}

return RedirectToAction("AdminPanel");

}

[Authorize]

public async Task<IActionResult> AddArticle()

{

ArticleModel newArticle = new ArticleModel();

return View(newArticle);

}

[HttpPost]

[Authorize]

public async Task<IActionResult> AddArticle(ArticleModel newArticle)

{

if (ModelState.IsValid)

{

var existingArticle = await \_context.Articles.FirstOrDefaultAsync(a => a.Title == newArticle.Title);

if (existingArticle != null)

{

ModelState.AddModelError(string.Empty, "Сттатя з таким заголовком вже існує!");

return View(newArticle);

}

var article = new ArticleModel { Title = newArticle.Title, Description = newArticle.Description };

await \_context.Articles.AddAsync(article);

await \_context.SaveChangesAsync();

return RedirectToAction("AdminPanel");

}

return View(newArticle);

}

[HttpPost]

[Authorize]

public async Task<IActionResult> UpdateArticle(int articleId, string Title, string Description)

{

var article = await \_context.Articles.FirstOrDefaultAsync(a => a.Id == articleId);

article.Description = Description;

if (article != null)

{

article.Title = Title;

article.Description = Description;

\_context.Update(article);

await \_context.SaveChangesAsync();

return RedirectToAction("AdminPanel");

}

return RedirectToAction("AdminPanel");

}

[HttpPost]

[Authorize]

public async Task<IActionResult> DeleteArticle(int articleId)

{

var article = await \_context.Articles.FindAsync(articleId);

if (article != null)

{

\_context.Remove(article);

await \_context.SaveChangesAsync();

return RedirectToAction("AdminPanel");

}

return RedirectToAction("AdminPanel");

}

}

}

Клас «HomeContlroller»

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ProjectDelivery.Models;

using System.Diagnostics;

namespace ProjectDelivery.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Index(TrackingModel trckmd)

{

if (ModelState.IsValid)

{

return RedirectToAction("Tracking", "Service", new { trackingModel = trckmd });

}

return View(trckmd);

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

Клас «ServiceContlroller»

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using ProjectDelivery.Classes;

using ProjectDelivery.Data;

using ProjectDelivery.Enums;

using ProjectDelivery.Models;

using System.Security.Claims;

namespace ProjectDelivery.Controllers

{

public class ServiceController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly UserManager<AccModel> \_userManager;

public ServiceController(ApplicationDbContext context, UserManager<AccModel> userManager)

{

\_context = context;

\_userManager = userManager;

}

public async Task<IActionResult> Index()

{

return View();

}

public async Task<IActionResult> Tracking(int id)

{

TrackingModel = new TrackingModel();

trackingModel.Id = id;

return View(trackingModel);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Tracking(TrackingModel trackingModel)

{

if (ModelState.IsValid)

{

var info = \_context.Packages.FirstOrDefault(p => p.Id == trackingModel.Id);

if (info != null)

{

trackingModel.Name = info.recipientName;

trackingModel.City = info.City.ToString();

trackingModel.Description = info.Description;

trackingModel.IsFind = true;

return View(trackingModel);

}

else

{

trackingModel.IsFind = false;

trackingModel.Text = "Не знайдено!";

return View(trackingModel);

}

}

return View(trackingModel);

}

public async Task<IActionResult> Calculate(int w, int c, EnumCities city)

{

CalcModel model = new CalcModel

{

Weight = w,

Cost = c,

City = city

};

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Calculate(CalcModel model)

{

if (ModelState.IsValid)

{

double w = model.Weight;

double c = model.Cost;

int price = 80;

if (w > 5 && w <= 10)

price = 90;

if (w > 10 && w <= 25)

price = 120;

model.Price = c \* 0.01 + price;

View(model);

}

return View(model);

}

public async Task<IActionResult> Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

[Authorize]

public async Task<IActionResult> Create(Package package)

{

if (ModelState.IsValid)

{

var recipient = await new FindingUser().FindByPhoneNumberAsync(\_userManager, package.Phone);

string currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

if (recipient != null && currentUserId != recipient.Id)

{

package.RecipientId = recipient.Id;

package.recipientName = recipient.Name;

}

if (currentUserId != null)

{

package.SenderId = currentUserId;

var user = await \_userManager.GetUserAsync(User);

package.senderName = user?.Name;

\_context.Packages.Add(package);

await \_context.SaveChangesAsync();

return RedirectToAction("Index", "Home");

}

}

return View(package);

}

public async Task<IActionResult> Help()

{

List<ArticleModel> model = \_context.Articles.ToList();

return View(model);

}

}

}