Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Комп’ютерних наук та інформаційних технологій"

**Звіт**

**Лабораторна робота 5**

з дисципліни "Проєктування серверних застосунків"

Виконав: студент групи КН-922В

Мельник Б. В.

Перевірив:

Івашко А. В.

Харків 2025

**Зміст**

[Завдання 1 4](#_Toc194615511)

[Текст програми 4](#_Toc194615512)

[Результати роботи програми 14](#_Toc194615513)

[Алгоритм побудови кодів 16](#_Toc194615514)

[Функціональність 17](#_Toc194615515)

[Висновок 17](#_Toc194615516)

[\*Завдання 2 18](#_Toc194615517)

[Текст програми 18](#_Toc194615518)

[Результати роботи програми 26](#_Toc194615519)

[Алгоритм побудови кодів 27](#_Toc194615520)

[Функціональність 28](#_Toc194615521)

[Висновок 28](#_Toc194615522)

[\*Завдання 3 29](#_Toc194615523)

[Текст програми 29](#_Toc194615524)

[Результати роботи програми 36](#_Toc194615525)

[Алгоритм побудови кодів 37](#_Toc194615526)

[Функціональність 38](#_Toc194615527)

[Висновок 39](#_Toc194615528)

[Завдання 4 40](#_Toc194615529)

[Текст програми 40](#_Toc194615530)

[Результати роботи програми 51](#_Toc194615531)

[Алгоритм побудови кодів 52](#_Toc194615532)

[Функціональність 53](#_Toc194615533)

[Висновок 54](#_Toc194615534)

[\*Завдання 5 55](#_Toc194615535)

[Текст програми 55](#_Toc194615536)

[Результати роботи програми 63](#_Toc194615537)

[Алгоритм побудови кодів 65](#_Toc194615538)

[Функціональність 66](#_Toc194615539)

[Висновок 67](#_Toc194615540)

[\*Завдання 6 68](#_Toc194615541)

[Текст програми 68](#_Toc194615542)

[Результати роботи програми 76](#_Toc194615543)

[Алгоритм побудови кодів 78](#_Toc194615544)

[Функціональність 79](#_Toc194615545)

[Висновок 79](#_Toc194615546)

[Завдання 7 80](#_Toc194615547)

[Текст програми 80](#_Toc194615548)

[Результати роботи програми 90](#_Toc194615549)

[Алгоритм побудови кодів 91](#_Toc194615550)

[Функціональність 92](#_Toc194615551)

[Висновок 92](#_Toc194615552)

[\*Завдання 8 94](#_Toc194615553)

[Текст програми 94](#_Toc194615554)

[Результати роботи програми 103](#_Toc194615555)

[Алгоритм побудови кодів 104](#_Toc194615556)

[Функціональність 105](#_Toc194615557)

[Висновок 106](#_Toc194615558)

[\*Завдання 9 107](#_Toc194615559)

[Текст програми 107](#_Toc194615560)

[Результати роботи програми 116](#_Toc194615561)

[Алгоритм побудови кодів 117](#_Toc194615562)

[Функціональність 118](#_Toc194615563)

[Висновок 119](#_Toc194615564)

[Загальний Висновок 120](#_Toc194615565)

**Тема:** Використання рівнів доступності (scopes) даних для обміну інформацією під час роботи MVC WEB-застосунку.

**Мета роботи:** : ознайомитися із різними варіантами визначення даних щодо рівня їхньої досяжності та способами використання для обміну інформацією у контролерах та поданнях WEB-застосунків.

**Програмні засоби розробки:** середовище програмування (IDE-Integrated De-velopment Environment) MS Visual Studio, програмний фреймворк ASP.NET версії не нижче 6.0 та дуже бажано використання дизайнерського фреймво-рку BOOTSTRAP, або якогось іншого відповідного аналогу, MySQL Server, або якась аналогічна СУБД.

**Індивідуальні завдання**

**Варіант №12**

# **Завдання 1**

Створити зв'язку ControllerlView1 Controller2-View2.- Controller3 --View3 Controller4 - View4 Де View1, View2, View3 задають відповідні сторони трикутника Controller2, Controller3, Controller4 зберігають значення сторін у потрібному «скопі» досяжності та реалізують навігацію між сторінками. На View4 розраховується параметр трикутника згідно власного варіанту.

|  |  |
| --- | --- |
| **№ варіанту** | **Функція розрахунку параметру трикутника** |
| 12 | Визначити наявність чи відсутність тупого кута у трикутнику |

## **Текст програми:**

**Controllers/Controller1.cs:**

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Controller1 : Controller

{

// GET: Controller1/View1

[HttpGet]

public IActionResult View1()

{

// Відображаємо сторінку для введення сторони A

return View();

}

// POST: Controller1/View1

[HttpPost]

public IActionResult View1(string sideA)

{

// Зберігаємо значення сторони A для використання в наступних кроках

TempData["SideA"] = sideA;

// Перехід до введення сторони B у наступному контролері

return RedirectToAction("View2", "Controller2");

}

}

}

**Controllers/Controller2.cs:**

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Controller2 : Controller

{

// GET: Controller2/View2

[HttpGet]

public IActionResult View2()

{

// Перевірка: якщо сторона A не введена, повертаємося на першу сторінку

if (TempData["SideA"] == null)

{

return RedirectToAction("View1", "Controller1");

}

// Зберігаємо значення сторони A для наступних запитів

TempData.Keep("SideA");

return View();

}

// POST: Controller2/View2

[HttpPost]

public IActionResult View2(string sideB)

{

// Записуємо значення сторони B у сесію

HttpContext.Session.SetString("SideB", sideB);

// Перехід до наступного введення (сторона C)

return RedirectToAction("View3", "Controller3");

}

}

}

**Controllers/Controller3.cs:**

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Controller3 : Controller

{

// GET: Controller3/View3

[HttpGet]

public IActionResult View3()

{

// Переконуємось, що дані сторін A та B вже введені

if (TempData["SideA"] == null || HttpContext.Session.GetString("SideB") == null)

{

return RedirectToAction("View1", "Controller1");

}

// Залишаємо дані сторони A для подальшого використання

TempData.Keep("SideA");

return View();

}

// POST: Controller3/View3

[HttpPost]

public IActionResult View3(string sideC)

{

// Передаємо значення сторони C через параметри запиту до наступного контролера

return RedirectToAction("View4", "Controller4", new { sideC = sideC });

}

}

}

**Controllers/Controller4.cs:**

using Microsoft.AspNetCore.Mvc;

using System;

namespace Task\_1.Controllers

{

public class Controller4 : Controller

{

// GET: Controller4/View4?sideC=...

[HttpGet]

public IActionResult View4(string sideC)

{

// Перевірка: чи є всі необхідні дані для обчислення

if (TempData["SideA"] == null || HttpContext.Session.GetString("SideB") == null || string.IsNullOrEmpty(sideC))

{

return RedirectToAction("View1", "Controller1");

}

// Спроба перетворення введених даних у числа

if (!double.TryParse(TempData["SideA"].ToString(), out double a))

{

a = 0; // В разі помилки встановлюємо значення 0

}

if (!double.TryParse(HttpContext.Session.GetString("SideB"), out double b))

{

b = 0;

}

if (!double.TryParse(sideC, out double c))

{

c = 0;

}

// Перевірка чи введені значення утворюють коректний трикутник

if (a <= 0 || b <= 0 || c <= 0 || (a + b <= c) || (a + c <= b) || (b + c <= a))

{

ViewBag.Result = "Введені значення не утворюють трикутник.";

return View();

}

// Обчислення: визначаємо чи є тупий кут у трикутнику

// Знаходимо найдовшу сторону

double longest = Math.Max(a, Math.Max(b, c));

double sumSquares = 0;

if (Math.Abs(longest - a) < 0.0001)

sumSquares = b \* b + c \* c;

else if (Math.Abs(longest - b) < 0.0001)

sumSquares = a \* a + c \* c;

else

sumSquares = a \* a + b \* b;

// Якщо квадрат найдовшої сторони більший за суму квадратів інших, трикутник має тупий кут

bool isObtuse = (longest \* longest > sumSquares);

string resultText = isObtuse ? "трикутник має тупий кут." : "Тупий кут відстутній у трикутнику.";

ViewBag.Result = $"Для трикутника зі сторонами: a = {a}, b = {b}, c = {c} => {resultText}";

return View();

}

}

}

**Views/Controller1/View1.cshtml:**

@{

ViewData["Title"] = "Введення сторони A";

// Встановлюємо клас для фонового зображення на сторінці

ViewBag.BodyClass = "bg1";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<h2 class="text-center">Введення сторони A</h2>

<form asp-action="View1" method="post">

<div class="form-group">

<label for="sideA">Сторона A:</label>

<!-- Поле для введення значення сторони A -->

<input type="text" class="form-control" id="sideA" name="sideA" placeholder="Введіть довжину сторони A" required />

</div>

<!-- Кнопка для переходу до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Views/Controller2/View2.cshtml:**

@{

ViewData["Title"] = "Введення сторони B";

// Використовуємо інший клас для фону, щоб відрізнити сторінки

ViewBag.BodyClass = "bg2";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<h2 class="text-center">Введення сторони B</h2>

<form asp-action="View2" method="post">

<div class="form-group">

<label for="sideB">Сторона B:</label>

<!-- Введення значення для сторони B -->

<input type="text" class="form-control" id="sideB" name="sideB" placeholder="Введіть довжину сторони B" required />

</div>

<!-- Переходимо до наступного введення -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Views/Controller3/View3.cshtml:**

@{

ViewData["Title"] = "Введення сторони C";

// Задаємо клас для фону сторінки з введенням третьої сторони

ViewBag.BodyClass = "bg3";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<h2 class="text-center">Введення сторони C</h2>

<form asp-action="View3" method="post">

<div class="form-group">

<label for="sideC">Сторона C:</label>

<!-- Поле для введення сторони C -->

<input type="text" class="form-control" id="sideC" name="sideC" placeholder="Введіть довжину сторони C" required />

</div>

<!-- Кнопка для обчислення результату -->

<button type="submit" class="btn btn-primary btn-block">Розрахувати</button>

</form>

</div>

</div>

</div>

**Views/Controller4/View4.cshtml:**

@{

ViewData["Title"] = "Результат";

// Використовуємо клас для сторінки з результатом

ViewBag.BodyClass = "bg4";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box text-center">

<h2>Результат</h2>

<!-- Відображення обчисленого результату -->

<p>@ViewBag.Result</p>

<!-- Посилання для запуску нового обчислення -->

<a href="@Url.Action("View1", "Controller1")" class="btn btn-primary">Нове обчислення</a>

</div>

</div>

</div>

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>@ViewData["Title"] - Завдання 1 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap (CDN) -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<**link** rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("View1", "Controller1")">Завдання 1 (Lab 5)</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" href="@Url.Action("View1", "Controller1")">Сторона A</a>

</li>

<li class="nav-item">

<a class="nav-link" href="@Url.Action("View2", "Controller2")">Сторона B</a>

</li>

<li class="nav-item">

<a class="nav-link" href="@Url.Action("View3", "Controller3")">Сторона C</a>

</li>

</ul>

</div>

</nav>

<!-- Основний контейнер для вмісту -->

<main class="container">

@RenderBody()

</main>

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Загальні стилі для повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використання flex для розташування елементів на сторінці \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає увесь доступний простір \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди розташовується внизу \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних сторінок \*/

body.bg1 {

/\* Фон для першої сторінки \*/

background: url('https://img.freepik.com/free-photo/morskie-oko-tatry\_1204-510.jpg?t=st=1743410338~exp=1743413938~hmac=1ab8e678fda2eb7170d5ca79c62e0b981815b571f12622676b8d6b77a81423cd&w=1380') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для другої сторінки \*/

background: url('https://img.freepik.com/free-photo/vestrahorn-mountains-stokksnes-iceland\_335224-667.jpg?t=st=1743412003~exp=1743415603~hmac=12e2ecd58a8dea89f7b89fbc85c954368b412f4c77180aca043ef1361625b730&w=1380') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для третьої сторінки \*/

background: url('https://img.freepik.com/free-photo/beautiful-shot-natural-scenery-autumn\_181624-25934.jpg?t=st=1743412524~exp=1743416124~hmac=9d9f5eb2d50b007e8b811a1d85ed4f6e69bb5ecbaa8f42e0a7d9f4b764585638&w=1380') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для сторінки з результатом \*/

background: url('https://img.freepik.com/free-photo/scenic-view-mountains-lake\_53876-138187.jpg?t=st=1743412542~exp=1743416142~hmac=2d0c5f47c5f58f05e66b05ee8dbe68f8697c85d50a0b22084452025e24e96521&w=1380') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація форми: прозорий блок з анімацією \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9); /\* напівпрозорий фон для читабельності \*/

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px); /\* трохи піднімаємо елемент \*/

}

to {

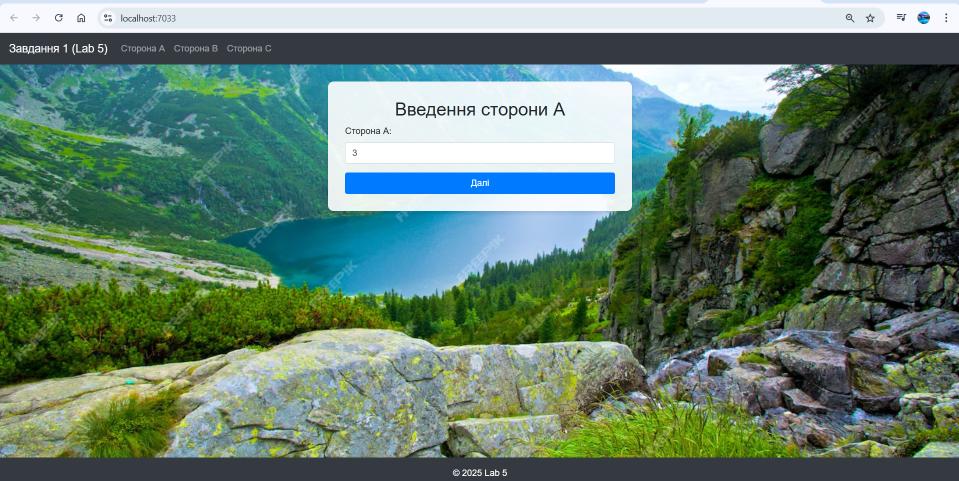
opacity: 1;

transform: translateY(0); /\* повертаємо на місце \*/

}

}

## **Результати роботи програми:**



Изображение выглядит как облако, текст, снимок экрана, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, дерево, вода, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, дерево, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **Створення контролерів**:
   * Кожен контролер відповідає за одну з чотирьох сторінок (View).
   * Кожен контролер обробляє GET та POST запити для введення сторін трикутника.
   * Дані сторін передаються між контролерами за допомогою TempData і Session.
2. **Контролери**:
   * Controller1: Відповідає за введення сторони A. Після введення перенаправляє до Controller2.
   * Controller2: Відповідає за введення сторони B. Зберігає значення сторони в сесії і перенаправляє до Controller3.
   * Controller3: Відповідає за введення сторони C. Після введення перенаправляє до Controller4, де розраховується параметр трикутника.
   * Controller4: Перевіряє, чи можна побудувати трикутник за введеними даними, і визначає, чи має трикутник тупий кут. Результат виводиться на сторінці.
3. **Алгоритм визначення тупого кута**:
   * Знаходимо найдовшу сторону трикутника.
   * Перевіряємо, чи квадрат найдовшої сторони більший за суму квадратів двох інших сторін.
   * Якщо так, то трикутник має тупий кут.
4. **Погляди (Views)**:
   * Для кожного контролера створюються відповідні сторінки, де користувач може ввести довжини сторін трикутника.
   * Для сторінки з результатом виводиться інформація про те, чи є тупий кут у трикутнику, або вказується, що введені дані не утворюють трикутник.

## **Функціональність**

* **Перехід між сторінками**: Кожна сторінка дозволяє користувачу ввести одну зі сторін трикутника та передати значення на наступну сторінку.
* **Валідація вводу**: На сторінці Controller4 перевіряється, чи утворюють введені значення трикутник. Якщо ні, виводиться повідомлення про помилку.
* **Обчислення параметра**: У Controller4 розраховується наявність тупого кута. Якщо квадрат найдовшої сторони більше за суму квадратів двох інших сторін, то трикутник має тупий кут.

## **Висновок**

Завдання передбачає реалізацію веб-додатку, що дозволяє користувачу вводити довжини сторін трикутника через послідовні сторінки. У результаті програма обчислює, чи є у трикутника тупий кут. Реалізована система перевіряє коректність введених даних і надає відповідний результат. Система є інтуїтивно зрозумілою, а структура проекту добре організована, що дозволяє легко додавати нові функціональні можливості в майбутньому.

# **\*Завдання 2**

Виконати минуле завдання за допомогою однієї View яка змінює інтер-фейс подання в залежності від кількості введених сторін та одного Controller, який зберігає чергову сторону трикутника чи розраховує його параметр теж в залежності від кількості введених сторін.

## **Текст програми:**

**Index.cshtml:**

@{

// Визначаємо, на якому кроці перебуваємо і яку сторону вводити

int step = ViewBag.Step;

string sideName = "";

switch (step)

{

case 1: sideName = "A"; break;

case 2: sideName = "B"; break;

case 3: sideName = "C"; break;

}

// Якщо всі сторони не введено – показуємо форму, інакше заголовок "Результат"

ViewData["Title"] = step < 4 ? $"Введення сторони {sideName}" : "Результат";

}

@if (step < 4)

{

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок для форми вводу -->

<h2 class="text-center">Введення сторони @sideName</h2>

<form asp-action="Index" method="post">

<div class="form-group">

<!-- Підпис до поля введення для сторони -->

<label for="sideInput">Сторона @sideName:</label>

<input type="text" class="form-control" id="sideInput" name="sideInput" placeholder="Введіть довжину сторони @sideName" required />

</div>

<!-- Кнопка для переходу до наступного вводу -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

}

else

{

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box text-center">

<!-- Відображення результату розрахунку -->

<h2>Результат</h2>

<p>@ViewBag.Result</p>

<!-- Посилання для початку нового обчислення -->

<a href="@Url.Action("Index", "Triangle", new { reset = true })" class="btn btn-primary">Нове обчислення</a>

</div>

</div>

</div>

}

**Controllers/ TriangleController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

namespace Task\_2.Controllers

{

public class TriangleController : Controller

{

// GET: Triangle/Index

[HttpGet]

public IActionResult Index(bool reset = false)

{

// Якщо користувач вирішив почати спочатку, очищаємо сесію

if (reset)

{

HttpContext.Session.Remove("SideA");

HttpContext.Session.Remove("SideB");

HttpContext.Session.Remove("SideC");

}

// Отримуємо збережені дані введення сторін з сесії

var sideA = HttpContext.Session.GetString("SideA");

var sideB = HttpContext.Session.GetString("SideB");

var sideC = HttpContext.Session.GetString("SideC");

// Визначаємо, який крок зараз: якщо ще нічого не введено – перший крок,

// якщо одна чи дві сторони – відповідний крок, інакше переходимо до розрахунку

int step = 1;

if (string.IsNullOrEmpty(sideA))

step = 1;

else if (string.IsNullOrEmpty(sideB))

step = 2;

else if (string.IsNullOrEmpty(sideC))

step = 3;

else

step = 4; // Всі дані введено, можна проводити обчислення

ViewBag.Step = step;

// Змінюємо фон сторінки відповідно до поточного кроку

switch (step)

{

case 1: ViewBag.BodyClass = "bg1"; break;

case 2: ViewBag.BodyClass = "bg2"; break;

case 3: ViewBag.BodyClass = "bg3"; break;

case 4: ViewBag.BodyClass = "bg4"; break;

}

if (step == 4)

{

// Пробуємо перетворити введені рядки у числа

if (!double.TryParse(sideA, out double a))

a = 0;

if (!double.TryParse(sideB, out double b))

b = 0;

if (!double.TryParse(sideC, out double c))

c = 0;

// Перевірка на можливість утворення трикутника

if (a <= 0 || b <= 0 || c <= 0 || (a + b <= c) || (a + c <= b) || (b + c <= a))

{

ViewBag.Result = "Введені значення не утворюють трикутник.";

}

else

{

// Розрахунок: перевіряємо чи має трикутник тупий кут

double longest = Math.Max(a, Math.Max(b, c));

double sumSquares = 0;

if (Math.Abs(longest - a) < 0.0001)

sumSquares = b \* b + c \* c;

else if (Math.Abs(longest - b) < 0.0001)

sumSquares = a \* a + c \* c;

else

sumSquares = a \* a + b \* b;

bool isObtuse = (longest \* longest > sumSquares);

string resultText = isObtuse ? "трикутник має тупий кут." : "тупий кут відсутній у трикутнику.";

ViewBag.Result = $"Для трикутника зі сторонами: a = {a}, b = {b}, c = {c} => {resultText}";

}

}

return View();

}

// POST: Triangle/Index

[HttpPost]

public IActionResult Index(string sideInput)

{

// Отримуємо поточні значення сторін із сесії

var sideA = HttpContext.Session.GetString("SideA");

var sideB = HttpContext.Session.GetString("SideB");

if (string.IsNullOrEmpty(sideA))

{

// Якщо ще не введено першу сторону – записуємо в SideA

HttpContext.Session.SetString("SideA", sideInput);

}

else if (string.IsNullOrEmpty(sideB))

{

// Якщо перша сторона вже введена, записуємо другу сторону

HttpContext.Session.SetString("SideB", sideInput);

}

else

{

// Якщо введені перша і друга сторони – записуємо третю

HttpContext.Session.SetString("SideC", sideInput);

}

// Перенаправляємо користувача назад на Index для оновлення кроку

return RedirectToAction("Index");

}

}

}

**Models/ TriangleViewModel.cs:**

namespace Task\_2.Models

{

public class TriangleViewModel

{

// Крок введення:

// 0 – введення сторони A,

// 1 – введення сторони B,

// 2 – введення сторони C,

// 3 – показ результату

public int Step { get; set; }

// Зберігаємо введене значення сторони A

public string SideA { get; set; }

// Зберігаємо введене значення сторони B

public string SideB { get; set; }

// Зберігаємо введене значення сторони C

public string SideC { get; set; }

// Результат обчислення (повідомлення для користувача)

public string Result { get; set; }

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>@ViewData["Title"] - Завдання 2 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap (CDN) -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Triangle")">Завдання 2 (Lab 5)</a>

</nav>

<!-- Основний контейнер для вмісту -->

<main class="container">

@RenderBody()

</main>

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Забезпечення повного охоплення екрана \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використовуємо flex для організації контенту на сторінці \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер – займає весь доступний простір \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди залишається внизу \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних кроків \*/

body.bg1 {

/\* Фон для першої сторінки \*/

background: url('https://img.freepik.com/free-photo/morskie-oko-tatry\_1204-510.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для другої сторінки \*/

background: url('https://img.freepik.com/free-photo/vestrahorn-mountains-stokksnes-iceland\_335224-667.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для третьої сторінки \*/

background: url('https://img.freepik.com/free-photo/beautiful-shot-natural-scenery-autumn\_181624-25934.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для сторінки з результатом \*/

background: url('https://img.freepik.com/free-photo/scenic-view-mountains-lake\_53876-138187.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми з легкою прозорістю \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Проста анімація появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как облако, гора, снимок экрана, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, дерево, снимок экрана, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, дикая незаселенная местность, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **Контролер TriangleController**:
   * У цьому контролері відбувається обробка всіх запитів користувача.
   * **GET метод**: Перевіряється поточний стан введення сторін трикутника (чи введено сторони A, B і C). В залежності від кількості введених сторін визначається поточний крок (від 1 до 4). Якщо всі сторони введено, виконується перевірка на коректність трикутника та розрахунок наявності тупого кута.
   * **POST метод**: Користувач вводить одну зі сторін трикутника. Відповідно до поточного кроку значення зберігається в сесії. Після цього контролер перенаправляє на той самий крок для оновлення інтерфейсу.
2. **Index.cshtml (Представлення)**:
   * Ця сторінка відображає інтерфейс, який змінюється в залежності від поточного кроку.
   * Для кожного кроку користувачу пропонується ввести відповідну сторону трикутника.
   * Після введення всіх трьох сторін з'являється результат, де вказується, чи має трикутник тупий кут.
3. **Перевірка введених даних**:
   * У контролері перевіряється, чи можуть введені сторони утворити коректний трикутник.
   * Якщо умови трикутної нерівності не виконуються, користувач отримує повідомлення про помилку.
   * Якщо трикутник коректний, виконується розрахунок, чи є в ньому тупий кут.
4. **Розрахунок наявності тупого кута**:
   * Використовується стандартний метод для визначення тупого кута за допомогою теореми Піфагора: якщо квадрат найдовшої сторони більший за суму квадратів двох інших, трикутник має тупий кут.

## **Функціональність**

* **Інтерактивне введення сторін**: Користувач може по черзі ввести сторони трикутника через одну форму на одній сторінці. Кожен новий ввід зберігається в сесії.
* **Зміна інтерфейсу**: Інтерфейс автоматично змінюється в залежності від того, скільки сторін введено, що дозволяє користувачеві зрозуміти, на якому етапі він знаходиться.
* **Результат обчислення**: Після введення всіх трьох сторін програма визначає, чи можна побудувати трикутник, та чи має він тупий кут. Результат виводиться на тій же сторінці.

## **Висновок**

Завдання дозволяє реалізувати інтерфейс для введення сторін трикутника в одному місці, з динамічними змінами на основі введених даних. Використання сесій для зберігання введених сторін спрощує структуру програми, адже дозволяє уникнути необхідності передавати дані між різними контролерами. Програма коректно перевіряє можливість утворення трикутника, а також визначає, чи має він тупий кут, що робить її практичною для навчальних цілей.

# **\*Завдання 3**

Виконати минуле завдання, використовуючи лише один метод в контро-лері.

## **Текст програми:**

**Index.cshtml:**

@{

// Отримуємо номер кроку з ViewBag (це число, що визначає, яку сторону потрібно ввести)

int step = ViewBag.Step;

string sideName = "";

// В залежності від кроку визначаємо, яку сторону просимо ввести (A, B чи C)

switch (step)

{

case 1: sideName = "A"; break;

case 2: sideName = "B"; break;

case 3: sideName = "C"; break;

}

// Заголовок сторінки: якщо ще вводимо сторону – показуємо її, інакше результат

ViewData["Title"] = step < 4 ? $"Введення сторони {sideName}" : "Результат";

}

@if (step < 4)

{

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Виводимо заголовок форми вводу -->

<h2 class="text-center">Введення сторони @sideName</h2>

<form asp-action="Index" method="post">

<div class="form-group">

<!-- Підказка до поля вводу -->

<label for="sideInput">Сторона @sideName:</label>

<input type="text" class="form-control" id="sideInput" name="sideInput" placeholder="Введіть довжину сторони @sideName" required />

</div>

<!-- Кнопка для переходу до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

}

else

{

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box text-center">

<!-- Показуємо остаточний результат обчислень -->

<h2>Результат</h2>

<p>@ViewBag.Result</p>

<!-- Посилання для запуску нового обчислення -->

<a href="@Url.Action("Index", "Triangle", new { reset = true })" class="btn btn-primary">Нове обчислення</a>

</div>

</div>

</div>

}

**TriangleController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

namespace Task\_3.Controllers

{

public class TriangleController : Controller

{

// Єдиний метод, який обробляє і GET, і POST запити на одну дію

public IActionResult Index(string sideInput, bool reset = false)

{

// Якщо користувач натиснув "Нове обчислення" – чистимо збережені значення

if (reset)

{

HttpContext.Session.Remove("SideA");

HttpContext.Session.Remove("SideB");

HttpContext.Session.Remove("SideC");

}

// Якщо запит був методом POST, обробляємо введене значення

if (HttpContext.Request.Method.Equals("POST", StringComparison.OrdinalIgnoreCase))

{

var sideA = HttpContext.Session.GetString("SideA");

var sideB = HttpContext.Session.GetString("SideB");

if (string.IsNullOrEmpty(sideA))

{

// Перший ввід – записуємо як SideA

HttpContext.Session.SetString("SideA", sideInput);

}

else if (string.IsNullOrEmpty(sideB))

{

// Другий ввід – записуємо як SideB

HttpContext.Session.SetString("SideB", sideInput);

}

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("SideC")))

{

// Третій ввід – записуємо як SideC

HttpContext.Session.SetString("SideC", sideInput);

}

// Редірект, щоб уникнути повторного сабміту форми

return RedirectToAction("Index");

}

// Читаємо збережені значення сторін із сесії

var savedSideA = HttpContext.Session.GetString("SideA");

var savedSideB = HttpContext.Session.GetString("SideB");

var savedSideC = HttpContext.Session.GetString("SideC");

// Визначаємо поточний крок вводу залежно від того, які значення вже збережені

int step = 1;

if (string.IsNullOrEmpty(savedSideA))

step = 1;

else if (string.IsNullOrEmpty(savedSideB))

step = 2;

else if (string.IsNullOrEmpty(savedSideC))

step = 3;

else

step = 4; // Усі сторони введено – переходимо до розрахунку

ViewBag.Step = step;

// Призначаємо CSS-клас для body, щоб змінити фон відповідно до кроку

switch (step)

{

case 1: ViewBag.BodyClass = "bg1"; break;

case 2: ViewBag.BodyClass = "bg2"; break;

case 3: ViewBag.BodyClass = "bg3"; break;

case 4: ViewBag.BodyClass = "bg4"; break;

}

// Якщо всі три сторони введено, виконуємо обчислення трикутника

if (step == 4)

{

if (!double.TryParse(savedSideA, out double a))

a = 0;

if (!double.TryParse(savedSideB, out double b))

b = 0;

if (!double.TryParse(savedSideC, out double c))

c = 0;

// Перевірка на коректність даних – чи можуть вони утворити трикутник

if (a <= 0 || b <= 0 || c <= 0 || (a + b <= c) || (a + c <= b) || (b + c <= a))

{

ViewBag.Result = "Введені значення не утворюють трикутник.";

}

else

{

// Обчислюємо, чи має трикутник тупий кут

double longest = Math.Max(a, Math.Max(b, c));

double sumSquares = 0;

if (Math.Abs(longest - a) < 0.0001)

sumSquares = b \* b + c \* c;

else if (Math.Abs(longest - b) < 0.0001)

sumSquares = a \* a + c \* c;

else

sumSquares = a \* a + b \* b;

bool isObtuse = (longest \* longest > sumSquares);

string resultText = isObtuse ? "трикутник має тупий кут." : "тупий кут відсутній у трикутнику.";

ViewBag.Result = $"Для трикутника зі сторонами: a = {a}, b = {b}, c = {c} => {resultText}";

}

}

return View();

}

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Заголовок сторінки динамічно встановлюється, додаючи назву завдання -->

<title>@ViewData["Title"] - Завдання 3 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Простенька навігаційна панель -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Triangle")">Завдання 3 (Lab 5)</a>

</nav>

<!-- Основна частина вмісту сторінки -->

<main class="container">

@RenderBody()

</main>

<!-- Футер, який завжди залишається внизу -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення jQuery і Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Загальні налаштування сторінки для повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використовуємо flexbox для зручного розташування вмісту \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Головний контейнер займає всю доступну висоту \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди знаходиться внизу сторінки \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних кроків \*/

body.bg1 {

/\* Фон для кроку 1 \*/

background: url('https://img.freepik.com/free-photo/morskie-oko-tatry\_1204-510.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для кроку 2 \*/

background: url('https://img.freepik.com/free-photo/vestrahorn-mountains-stokksnes-iceland\_335224-667.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для кроку 3 \*/

background: url('https://img.freepik.com/free-photo/beautiful-shot-natural-scenery-autumn\_181624-25934.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для відображення результату \*/

background: url('https://img.freepik.com/free-photo/scenic-view-mountains-lake\_53876-138187.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми – прозорий фон з плавною анімацією \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація появи блоку форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

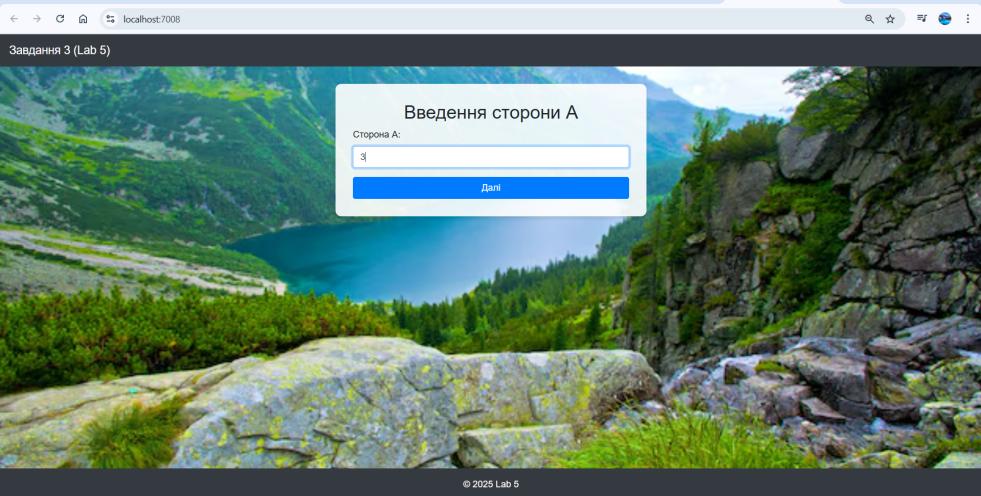
opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**



Изображение выглядит как облако, снимок экрана, гора, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, дерево, снимок экрана, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, снимок экрана, дикая незаселенная местность

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **Контролер TriangleController**:
   * В даному завданні використовується **один метод** для обробки і GET, і POST запитів.
   * **GET метод**: Перевіряє стан сесії для визначення поточного кроку (від 1 до 4). Кроки визначають, яку сторону трикутника користувач має ввести (A, B, або C). Якщо всі сторони введено, програма переходить до обчислення результату.
   * **POST метод**: Користувач вводить одну зі сторін трикутника. В залежності від поточного кроку значення зберігається в сесії. Після цього метод перенаправляє на той самий крок для оновлення інтерфейсу.
2. **Інтерфейс (Index.cshtml)**:
   * Змінюється в залежності від поточного кроку (зміна заголовка, тексту введення та фону сторінки).
   * Користувач вводить одну зі сторін, і на основі введеного значення сторони, програма змінює наступний крок.
   * Після введення всіх трьох сторін, користувач отримує результат обчислення: чи є тупий кут у трикутнику.
3. **Перевірка введених даних**:
   * Програма перевіряє, чи введені сторони можуть утворити коректний трикутник. Для цього використовується стандартна перевірка трикутної нерівності.
   * Якщо трикутник може бути побудований, програма визначає, чи має він тупий кут, використовуючи формулу Піфагора.
4. **Обчислення тупого кута**:
   * Після того, як всі сторони введені, програма визначає найдовшу сторону і порівнює квадрат цієї сторони з сумою квадратів інших двох сторін.
   * Якщо квадрат найдовшої сторони більший за суму квадратів інших сторін, то трикутник має тупий кут.

## **Функціональність**

* **Інтерактивне введення сторін**: Користувач вводить сторони трикутника по черзі, і інтерфейс оновлюється на кожному кроці.
* **Обчислення параметрів трикутника**: Після введення всіх сторін програма перевіряє можливість утворення трикутника, а також розраховує, чи є тупий кут у трикутнику.
* **Динамічне відображення інтерфейсу**: В залежності від поточного кроку, змінюється заголовок, текст форми і фон сторінки, що покращує взаємодію з користувачем.
* **Збереження введених даних**: Дані сторін зберігаються в сесії, що дозволяє зберігати введені значення між запитами.

## **Висновок**

Завдання дозволяє реалізувати інтерфейс для поетапного введення сторін трикутника за допомогою одного методу в контролері. Це дозволяє спростити код, уникнути дублювання логіки і зробити його більш компактним і зручним для підтримки. В результаті програма стає гнучкішою та зручною для користувача, з можливістю обчислення параметрів трикутника та перевірки наявності тупого кута, з можливістю почати нове обчислення.

# **Завдання 4**

Створити чотирьох етапний WEB-процес реєстрації абстрактного акаунту. На кожному етапі реалізується запит на інформацію згідно до власного ва-ріанту. Після останнього етапу інформація має бути записана до відповід-ної таблиці бази даних.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ варіанту** | **1 етап** | **2 етап** | **3 етап** | **4 етап** |
| 1 | 2 | 3 | 4 | 5 |
| 12 | Password | Email | Address | Login |

## **Текст програми:**

**SQL-запит:**

USE RegistrationDB;

-- Створення таблиці Accounts для збереження даних реєстрації

CREATE TABLE Accounts (

Id INT IDENTITY(1,1) PRIMARY KEY,

Password NVARCHAR(100) NOT NULL,

Email NVARCHAR(100) NOT NULL,

Address NVARCHAR(200) NOT NULL,

Login NVARCHAR(50) NOT NULL,

CreatedAt DATETIME DEFAULT GETDATE()

);

**RegistrationController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

namespace Task\_4.Controllers

{

public class RegistrationController : Controller

{

private readonly IConfiguration \_configuration;

public RegistrationController(IConfiguration configuration)

{

// Зберігаємо конфігурацію для роботи з рядком підключення

\_configuration = configuration;

}

// Step 1: Введення пароля

[HttpGet]

public IActionResult Step1()

{

// Просто повертаємо форму вводу пароля

return View();

}

[HttpPost]

public IActionResult Step1(string password)

{

// Якщо пароль не введено – повертаємо форму з помилкою

if (string.IsNullOrWhiteSpace(password))

{

ModelState.AddModelError("", "Будь ласка, введіть пароль.");

return View();

}

// Зберігаємо пароль у сесії

HttpContext.Session.SetString("Password", password);

return RedirectToAction("Step2");

}

// Step 2: Введення Email

[HttpGet]

public IActionResult Step2()

{

// Якщо пароль не заданий, перенаправляємо назад до Step1

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")))

return RedirectToAction("Step1");

return View();

}

[HttpPost]

public IActionResult Step2(string email)

{

// Перевіряємо, чи Email не пустий

if (string.IsNullOrWhiteSpace(email))

{

ModelState.AddModelError("", "Будь ласка, введіть Email.");

return View();

}

// Зберігаємо Email у сесії

HttpContext.Session.SetString("Email", email);

return RedirectToAction("Step3");

}

// Step 3: Введення адреси

[HttpGet]

public IActionResult Step3()

{

// Якщо пароль або Email не задані, повертаємося на початок

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")) ||

string.IsNullOrEmpty(HttpContext.Session.GetString("Email")))

return RedirectToAction("Step1");

return View();

}

[HttpPost]

public IActionResult Step3(string address)

{

// Перевірка на заповнення адреси

if (string.IsNullOrWhiteSpace(address))

{

ModelState.AddModelError("", "Будь ласка, введіть адресу.");

return View();

}

// Зберігаємо адресу у сесії

HttpContext.Session.SetString("Address", address);

return RedirectToAction("Step4");

}

// Step 4: Введення логіну та запис до бази даних

[HttpGet]

public IActionResult Step4()

{

// Якщо даних не вистачає – повертаємо користувача до початку

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")) ||

string.IsNullOrEmpty(HttpContext.Session.GetString("Email")) ||

string.IsNullOrEmpty(HttpContext.Session.GetString("Address")))

return RedirectToAction("Step1");

return View();

}

[HttpPost]

public IActionResult Step4(string login)

{

// Перевірка, чи логін введено

if (string.IsNullOrWhiteSpace(login))

{

ModelState.AddModelError("", "Будь ласка, введіть логін.");

return View();

}

// Зберігаємо логін у сесії

HttpContext.Session.SetString("Login", login);

// Збираємо всі дані для запису в базу

string password = HttpContext.Session.GetString("Password");

string email = HttpContext.Session.GetString("Email");

string address = HttpContext.Session.GetString("Address");

string loginVal = HttpContext.Session.GetString("Login");

// Рядок підключення до БД

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Відкриваємо з'єднання і записуємо дані в таблицю Accounts

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "INSERT INTO Accounts (Password, Email, Address, Login) VALUES (@Password, @Email, @Address, @Login)";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", password);

cmd.Parameters.AddWithValue("@Email", email);

cmd.Parameters.AddWithValue("@Address", address);

cmd.Parameters.AddWithValue("@Login", loginVal);

cmd.ExecuteNonQuery();

}

}

// Якщо запис успішний – зберігаємо повідомлення про успіх

TempData["Success"] = "Реєстрацію успішно завершено!";

}

catch (Exception ex)

{

// Якщо сталася помилка – записуємо повідомлення про помилку і залишаємо користувача на цій же сторінці

TempData["Error"] = "Сталася помилка при записі даних: " + ex.Message;

return RedirectToAction("Step4");

}

// Очищаємо сесію після завершення реєстрації

HttpContext.Session.Clear();

return RedirectToAction("Success");

}

// Success: Повідомлення про успішну реєстрацію

[HttpGet]

public IActionResult Success()

{

// Показуємо сторінку успіху

return View();

}

}

}

**Registration/ Step1.cshtml:**

@{

// Використовуємо стандартний макет сторінки

Layout = "~/Views/Shared/\_Layout.cshtml";

// Встановлюємо заголовок для цієї сторінки

ViewData["Title"] = "Введення пароля";

// Задаємо клас для body, щоб задати потрібний фон (bg1)

ViewBag.BodyClass = "bg1";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок форми -->

<h2 class="text-center">Введіть пароль</h2>

<form asp-action="Step1" method="post">

<div class="form-group">

<!-- Поле для введення пароля -->

<label for="password">Пароль:</label>

<input type="text" class="form-control" id="password" name="password" placeholder="Введіть пароль" required />

</div>

<!-- Кнопка для переходу до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Registration/ Step2.cshtml:**

@{

// Підключаємо макет

Layout = "~/Views/Shared/\_Layout.cshtml";

// Заголовок сторінки для введення Email

ViewData["Title"] = "Введення Email";

// Встановлюємо фон для цього кроку (bg2)

ViewBag.BodyClass = "bg2";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок для вводу Email -->

<h2 class="text-center">Введіть Email</h2>

<form asp-action="Step2" method="post">

<div class="form-group">

<!-- Поле введення для Email -->

<label for="email">Email:</label>

<input type="email" class="form-control" id="email" name="email" placeholder="Введіть Email" required />

</div>

<!-- Переходимо до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Registration/ Step3.cshtml:**

@{

// Встановлюємо макет і заголовок для введення адреси

Layout = "~/Views/Shared/\_Layout.cshtml";

ViewData["Title"] = "Введення адреси";

// Встановлюємо фон цього кроку (bg3)

ViewBag.BodyClass = "bg3";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок форми для адреси -->

<h2 class="text-center">Введіть адресу</h2>

<form asp-action="Step3" method="post">

<div class="form-group">

<!-- Поле для вводу адреси -->

<label for="address">Адреса:</label>

<input type="text" class="form-control" id="address" name="address" placeholder="Введіть адресу" required />

</div>

<!-- Переходимо до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Registration/ Step4.cshtml:**

@{

// Використовуємо основний макет

Layout = "~/Views/Shared/\_Layout.cshtml";

// Заголовок сторінки для введення логіну

ViewData["Title"] = "Введення логіну";

// Фон для цього кроку – bg4

ViewBag.BodyClass = "bg4";

// Отримуємо можливу помилку з TempData (якщо щось пішло не так)

var error = TempData["Error"] as string;

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок для вводу логіну -->

<h2 class="text-center">Введіть логін</h2>

@if (!string.IsNullOrEmpty(error))

{

<!-- Виводимо повідомлення про помилку, якщо є -->

<div class="alert alert-danger">@error</div>

}

<form asp-action="Step4" method="post">

<div class="form-group">

<!-- Поле введення логіну -->

<label for="login">Логін:</label>

<input type="text" class="form-control" id="login" name="login" placeholder="Введіть логін" required />

</div>

<!-- Кнопка для завершення реєстрації -->

<button type="submit" class="btn btn-primary btn-block">Зареєструвати</button>

</form>

</div>

</div>

</div>

**Registration/ Success.cshtml:**

@{

// Встановлюємо макет сторінки

Layout = "~/Views/Shared/\_Layout.cshtml";

// Заголовок, що повідомляє про успішну реєстрацію

ViewData["Title"] = "Реєстрація завершена";

// Використовуємо фон, аналогічний останньому кроку

ViewBag.BodyClass = "bg4";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box text-center">

<!-- Повідомлення про успішну реєстрацію -->

<h2>Реєстрацію успішно завершено!</h2>

<!-- Посилання для запуску нової реєстрації -->

<a href="@Url.Action("Step1", "Registration")" class="btn btn-primary">Нова реєстрація</a>

</div>

</div>

</div>

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Заголовок сторінки формується динамічно з ViewData -->

<title>@ViewData["Title"] - Завдання 4 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN для швидкого старту -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключення власних стилів -->

<**link** rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Верхня навігаційна панель -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Step1", "Registration")">

Завдання 4 (Lab 5) – Реєстрація

</a>

</nav>

<!-- Основний контейнер для динамічного вмісту -->

<main class="container">

@RenderBody()

</main>

<!-- Футер сторінки -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript файлів для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Забезпечення повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використовуємо flexbox для зручного розташування елементів \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає всю доступну висоту \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди знаходиться внизу сторінки \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних етапів реєстрації \*/

body.bg1 {

/\* Фон для кроку 1 \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-bhbe2xkopirx8goi.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для кроку 2 \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-7h76hbdcrtpawe3i.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для кроку 3 \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-3josvp4sieeo5ocn.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для кроку 4 та сторінки успіху \*/

background: url('https://i.imgur.com/HaJuteg.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми з легким прозорим фоном \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как дерево, небо, снимок экрана, вода

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, вода, пейзаж, гора

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, озеро, мультимедиа

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, текст, небо, на открытом воздухе

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как небо, снимок экрана, текст, на открытом воздухе

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит для створення таблиці**:
   * Спочатку створюється таблиця Accounts у базі даних для збереження даних користувачів. Це включає поля для пароля, електронної пошти, адреси, логіну, а також дату створення запису.
2. **Контролер RegistrationController**:
   * Контролер містить чотири етапи для реєстрації:
     + **Step1**: Крок для введення пароля. Якщо користувач не вводить пароль, з'являється помилка.
     + **Step2**: Крок для введення електронної пошти. Якщо електронна пошта не введена, показується помилка.
     + **Step3**: Крок для введення адреси. Якщо адреса не введена, з'являється помилка.
     + **Step4**: Крок для введення логіну та запису усіх даних у базу даних.
   * На кожному етапі, після введення даних, вони зберігаються в сесії для передачі між етапами.
3. **Метод Step4 (запис у базу даних)**:
   * Після завершення всіх етапів користувач вводить логін, і всі дані (пароль, електронна пошта, адреса, логін) записуються в таблицю Accounts у базі даних.
   * Якщо запис успішний, користувач отримує повідомлення про успішну реєстрацію. Якщо сталася помилка, виводиться повідомлення про помилку.
4. **Кроки реєстрації**:
   * **Step1**: Введення пароля.
   * **Step2**: Введення електронної пошти.
   * **Step3**: Введення адреси.
   * **Step4**: Введення логіну та завершення реєстрації.
5. **Успішне завершення**:
   * Після завершення реєстрації користувач бачить повідомлення про успішну реєстрацію і може розпочати нову реєстрацію.

## **Функціональність**

* **Чотири етапи реєстрації**: Користувач по черзі вводить пароль, електронну пошту, адресу і логін. На кожному етапі перевіряється правильність введених даних.
* **Зберігання даних у сесії**: Введені дані зберігаються у сесії між етапами, що дозволяє користувачу переходити між етапами без необхідності знову вводити попередню інформацію.
* **Запис у базу даних**: Після введення всіх даних система записує їх у таблицю Accounts бази даних.
* **Перевірка правильності введення**: Кожен етап перевіряється на наявність даних (пароль, електронна пошта, адреса, логін).
* **Успішне завершення реєстрації**: Після завершення реєстрації користувач отримує повідомлення про успіх і може почати процес знову.

## **Висновок**

Це завдання дозволяє створити чотирьохетапний процес реєстрації, який включає введення пароля, електронної пошти, адреси та логіну. Кожен етап перевіряється на коректність введених даних, а після завершення реєстрації всі дані зберігаються у базі даних. Використання сесії для зберігання введених даних між етапами спрощує взаємодію користувача з формою. Це завдання є корисним для вивчення процесів роботи з формами, сесіями та базами даних в ASP.NET Core.

# **\*Завдання 5**

Виконати минуле завдання за допомогою однієї View, яка змінює свій інтерфейс залежності від кількості введених даних майбутнього акаунту та одного Controller, який зберігає чергову порцію інформації акаунту в залежності від введеної інформації та додає її до відповідної таблиці.

## **Текст програми:**

**Index.csgtml:**

@{

// Підключаємо загальний макет сторінки

Layout = "~/Views/Shared/\_Layout.cshtml";

// Отримуємо номер поточного етапу з ViewBag (визначає, що вводити)

int step = ViewBag.Step;

// Заголовок форми – буде змінюватися залежно від етапу

string title = ViewBag.Title;

// Підказка для поля вводу (наприклад, "Пароль:", "Email:" тощо)

string prompt = ViewBag.Prompt;

// Отримуємо повідомлення про успіх, якщо реєстрація завершена

string success = ViewBag.Success as string;

// Отримуємо повідомлення про помилку, якщо щось пішло не так

string error = ViewBag.Error as string;

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

@if (!string.IsNullOrEmpty(success))

{

<!-- Якщо реєстрація пройшла успішно, виводимо повідомлення -->

<div class="alert alert-success text-center">

@success

</div>

<!-- Кнопка для запуску нового процесу реєстрації -->

<a href="@Url.Action("Index", "Registration", new { reset = true })" class="btn btn-primary btn-block">

Нова реєстрація

</a>

}

else

{

<!-- Якщо ще не завершено – показуємо заголовок форми -->

<h2 class="text-center">@title</h2>

@if (!string.IsNullOrEmpty(error))

{

<!-- Виводимо повідомлення про помилку, якщо воно є -->

<div class="alert alert-danger">

@error

</div>

}

<!-- Форма вводу для поточного етапу -->

<**form** **asp-action**="Index" method="post">

<div class="form-group">

<label for="input">@prompt</label>

<input type="text" class="form-control" id="input" name="input" placeholder="@prompt" required />

</div>

<!-- Кнопка для підтвердження введення і переходу до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</**form**>

}

</div>

</div>

</div>

**RegistrationController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

namespace Task\_5.Controllers

{

public class RegistrationController : Controller

{

private readonly IConfiguration \_configuration;

public RegistrationController(IConfiguration configuration)

{

// Зберігаємо об'єкт конфігурації для отримання рядка підключення до БД

\_configuration = configuration;

}

// GET: Registration/Index

[HttpGet]

public IActionResult Index(bool reset = false)

{

// Якщо користувач хоче почати з початку, очищаємо сесію та TempData

if (reset)

{

HttpContext.Session.Clear();

TempData["Error"] = null;

TempData["Success"] = null;

return RedirectToAction("Index");

}

// Визначаємо поточний етап заповнення форми за даними з сесії

int step = 1;

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")))

step = 1;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Email")))

step = 2;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Address")))

step = 3;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Login")))

step = 4;

ViewBag.Step = step;

// Призначаємо відповідний CSS-клас для фону залежно від етапу

switch (step)

{

case 1: ViewBag.BodyClass = "bg1"; break;

case 2: ViewBag.BodyClass = "bg2"; break;

case 3: ViewBag.BodyClass = "bg3"; break;

case 4: ViewBag.BodyClass = "bg4"; break;

}

// Встановлюємо заголовок і підказку для поля вводу в залежності від етапу

string title = "";

string prompt = "";

switch (step)

{

case 1:

title = "Введіть пароль";

prompt = "Пароль:";

break;

case 2:

title = "Введіть Email";

prompt = "Email:";

break;

case 3:

title = "Введіть адресу";

prompt = "Адреса:";

break;

case 4:

title = "Введіть логін";

prompt = "Логін:";

break;

}

// Передаємо дані у View через ViewBag

ViewBag.Title = title;

ViewBag.Prompt = prompt;

ViewBag.Error = TempData["Error"];

ViewBag.Success = TempData["Success"];

return View();

}

// POST: Registration/Index

[HttpPost]

public IActionResult Index(string input)

{

// Обробка даних в залежності від поточного етапу заповнення

// Етап 1: введення пароля

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")))

{

HttpContext.Session.SetString("Password", input);

}

// Етап 2: введення Email

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Email")))

{

HttpContext.Session.SetString("Email", input);

}

// Етап 3: введення адреси

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Address")))

{

HttpContext.Session.SetString("Address", input);

}

// Етап 4: введення логіну і запис до БД

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Login")))

{

HttpContext.Session.SetString("Login", input);

// Збираємо всі введені дані з сесії

string password = HttpContext.Session.GetString("Password");

string email = HttpContext.Session.GetString("Email");

string address = HttpContext.Session.GetString("Address");

string login = HttpContext.Session.GetString("Login");

// Отримуємо рядок підключення до БД з конфігурації

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Підключаємось до бази і записуємо дані в таблицю Accounts

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "INSERT INTO Accounts (Password, Email, Address, Login) VALUES (@Password, @Email, @Address, @Login)";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", password);

cmd.Parameters.AddWithValue("@Email", email);

cmd.Parameters.AddWithValue("@Address", address);

cmd.Parameters.AddWithValue("@Login", login);

cmd.ExecuteNonQuery();

}

}

// Якщо запис успішний – зберігаємо повідомлення для користувача

TempData["Success"] = "Реєстрацію успішно завершено!";

}

catch (Exception ex)

{

// Якщо сталася помилка – зберігаємо повідомлення про помилку

TempData["Error"] = "Сталася помилка при записі даних: " + ex.Message;

}

// Очищаємо сесію, щоб уникнути повторних записів

HttpContext.Session.Clear();

return RedirectToAction("Index");

}

// Редірект після кожного POST для запобігання повторного сабміту форми

return RedirectToAction("Index");

}

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Заголовок сторінки формується динамічно з ViewData -->

<title>@ViewData["Title"] - Завдання 5 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN для швидкого старту -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключаємо власні стилі -->

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Верхня навігаційна панель з брендингом -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Registration")">

Завдання 5 (Lab 5) – Реєстрація

</a>

</nav>

<!-- Основний контейнер для відображення вмісту конкретної сторінки -->

<main class="container">

@RenderBody()

</main>

<!-- Футер з інформацією про авторські права -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JS файлів для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Загальні налаштування: забезпечуємо повноекранне відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використання flexbox для організації розташування елементів \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Головний контейнер займає всю доступну висоту \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди залишається внизу сторінки \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для кожного етапу реєстрації \*/

body.bg1 {

/\* Фон для етапу 1 (введення пароля) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-bhbe2xkopirx8goi.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для етапу 2 (введення Email) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-7h76hbdcrtpawe3i.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для етапу 3 (введення адреси) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-3josvp4sieeo5ocn.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для етапу 4 (введення логіну та підсумок) \*/

background: url('https://i.imgur.com/HaJuteg.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми – прозорий фон для кращої читабельності \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Проста анімація появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как дерево, небо, вода, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, пейзаж, облако, вода

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, озеро, мультимедиа

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, небо, на открытом воздухе, земля

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как дерево, небо, снимок экрана, гора

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как программное обеспечение, Мультимедийное программное обеспечение, текст, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит**:
   * Створення таблиці Accounts для зберігання даних користувача в базі даних. Таблиця містить поля для пароля, email, адреси, логіну, а також дату створення.
2. **Контролер RegistrationController**:
   * **GET метод**:
     + Перевірка поточного етапу реєстрації (1 - пароль, 2 - email, 3 - адреса, 4 - логін).
     + Визначення етапу залежно від того, які дані вже введені.
     + Установка значень для заголовка та підказки відповідно до етапу.
     + Передача необхідних даних в View через ViewBag.
   * **POST метод**:
     + Користувач вводить одну з частин інформації (пароль, email, адресу, логін).
     + Дані зберігаються в сесії та передаються до наступного етапу.
     + Після введення всіх даних (логіну) дані записуються в базу даних.
     + У разі успішної реєстрації, з'являється повідомлення про успіх.
3. **Інтерфейс (View)**:
   * Єдина форма, що динамічно змінює свій інтерфейс в залежності від поточного етапу (пароль, email, адреса, логін).
   * Після кожного етапу дані зберігаються в сесії для переходу до наступного етапу.
   * Після завершення реєстрації користувач отримує повідомлення про успішну реєстрацію та може почати нову реєстрацію.
4. **Форма на кожному етапі**:
   * Для кожного етапу (пароль, email, адреса, логін) користувач вводить відповідну інформацію.
   * Форма змінюється залежно від того, на якому етапі перебуває користувач.
   * Після завершення етапу дані зберігаються в сесії, а потім переходять до наступного етапу.
5. **Запис до бази даних**:
   * Після введення всіх даних, вони записуються в таблицю Accounts у базі даних.
   * Якщо запис успішний, користувач отримує повідомлення про успіх. Якщо сталася помилка, виводиться повідомлення про помилку.

## **Функціональність**

* **Чотирьохетапний процес реєстрації**: Користувач по черзі вводить пароль, email, адресу та логін. На кожному етапі зберігаються відповідні дані в сесії.
* **Динамічна зміна інтерфейсу**: Інтерфейс змінюється залежно від поточного етапу, що дозволяє зручно вводити кожну частину даних.
* **Збереження даних у сесії**: Кожен етап зберігає введені дані в сесії, щоб користувач міг переходити до наступного етапу без повторного введення попередньої інформації.
* **Запис до бази даних**: Після завершення всіх етапів, система записує всі дані в таблицю Accounts у базі даних.
* **Перевірка на помилки**: Якщо під час реєстрації виникають помилки, вони відображаються користувачеві. У разі успіху з'являється повідомлення про завершення реєстрації.

## **Висновок**

Це завдання дозволяє створити інтерфейс для поетапної реєстрації користувачів, що зберігає введені дані в сесії. Реєстрація включає введення пароля, email, адреси та логіну, а також запис результатів у базу даних після завершення всіх етапів. Використання однієї View та одного Controller спрощує структуру програми і дозволяє зручно обробляти кожен етап реєстрації. Система також дозволяє обробляти помилки на кожному етапі і відображати відповідні повідомлення для користувача.

# **\*Завдання 6**

Виконати минуле завдання, використовуючи лише один метод в контро-лері.

## **Текст програми:**

**Index.cshtml:**

@{

// Використовуємо стандартний макет сторінки

Layout = "~/Views/Shared/\_Layout.cshtml";

// Отримуємо поточний етап заповнення форми з ViewBag

int step = ViewBag.Step;

// Заголовок, який відображатиметься на формі (наприклад, "Введіть пароль")

string title = ViewBag.Title;

// Текст підказки для поля вводу (наприклад, "Пароль:")

string prompt = ViewBag.Prompt;

// Повідомлення про успіх (якщо реєстрація вже завершена)

string success = ViewBag.Success as string;

// Повідомлення про помилку (якщо сталася якась помилка)

string error = ViewBag.Error as string;

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

@if (!string.IsNullOrEmpty(success))

{

<!-- Якщо реєстрація успішна, показуємо повідомлення і кнопку для нової реєстрації -->

<div class="alert alert-success text-center">

@success

</div>

<a href="@Url.Action("Index", "Registration", new { reset = true })" class="btn btn-primary btn-block">

Нова реєстрація

</a>

}

else

{

<!-- Виводимо заголовок для поточного етапу -->

<h2 class="text-center">@title</h2>

@if (!string.IsNullOrEmpty(error))

{

<!-- Якщо є помилка, відображаємо її тут -->

<div class="alert alert-danger">

@error

</div>

}

<!-- Форма вводу даних для поточного етапу -->

<form asp-action="Index" method="post">

<div class="form-group">

<label for="input">@prompt</label>

<input type="text" class="form-control" id="input" name="input" placeholder="@prompt" required />

</div>

<!-- Кнопка для відправлення форми і переходу до наступного етапу -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

}

</div>

</div>

</div>

**RegistrationController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

namespace Task\_6.Controllers

{

public class RegistrationController : Controller

{

private readonly IConfiguration \_configuration;

public RegistrationController(IConfiguration configuration)

{

// Зберігаємо конфігурацію для подальшого доступу до рядка підключення до БД

\_configuration = configuration;

}

// Єдиний метод, що обробляє як GET, так і POST запити для реєстрації

public IActionResult Index(string input, bool reset = false)

{

// Якщо користувач хоче почати з початку, очищаємо сесію та TempData

if (reset)

{

HttpContext.Session.Clear();

TempData["Error"] = null;

TempData["Success"] = null;

return RedirectToAction("Index");

}

// Якщо запит має метод POST, обробляємо введені дані за етапами

if (Request.Method.Equals("POST", StringComparison.OrdinalIgnoreCase))

{

// Етап 1: введення пароля

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")))

{

HttpContext.Session.SetString("Password", input);

}

// Етап 2: введення Email

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Email")))

{

HttpContext.Session.SetString("Email", input);

}

// Етап 3: введення адреси

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Address")))

{

HttpContext.Session.SetString("Address", input);

}

// Етап 4: введення логіну та запис до БД

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Login")))

{

HttpContext.Session.SetString("Login", input);

// Отримуємо всі збережені дані з сесії

string password = HttpContext.Session.GetString("Password");

string email = HttpContext.Session.GetString("Email");

string address = HttpContext.Session.GetString("Address");

string login = HttpContext.Session.GetString("Login");

// Отримуємо рядок підключення до бази даних з конфігурації

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Підключаємось до бази і виконуємо вставку даних у таблицю Accounts

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "INSERT INTO Accounts (Password, Email, Address, Login) VALUES (@Password, @Email, @Address, @Login)";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", password);

cmd.Parameters.AddWithValue("@Email", email);

cmd.Parameters.AddWithValue("@Address", address);

cmd.Parameters.AddWithValue("@Login", login);

cmd.ExecuteNonQuery();

}

}

// Якщо запис пройшов успішно, зберігаємо повідомлення для користувача

TempData["Success"] = "Реєстрацію успішно завершено!";

}

catch (Exception ex)

{

// У випадку помилки зберігаємо текст помилки в TempData

TempData["Error"] = "Сталася помилка при записі даних: " + ex.Message;

}

// Очищаємо сесію, щоб уникнути повторного сабміту

HttpContext.Session.Clear();

return RedirectToAction("Index");

}

// Після обробки даних виконуємо редірект для запобігання повторного сабміту форми

return RedirectToAction("Index");

}

// Якщо запит GET – визначаємо поточний етап заповнення форми за даними в сесії

int step = 1;

if (string.IsNullOrEmpty(HttpContext.Session.GetString("Password")))

step = 1;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Email")))

step = 2;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Address")))

step = 3;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("Login")))

step = 4;

ViewBag.Step = step;

// Призначаємо відповідний CSS-клас для фону залежно від етапу

switch (step)

{

case 1: ViewBag.BodyClass = "bg1"; break;

case 2: ViewBag.BodyClass = "bg2"; break;

case 3: ViewBag.BodyClass = "bg3"; break;

case 4: ViewBag.BodyClass = "bg4"; break;

}

// Встановлюємо заголовок і підказку для форми згідно з поточним етапом

string title = "";

string prompt = "";

switch (step)

{

case 1:

title = "Введіть пароль";

prompt = "Пароль:";

break;

case 2:

title = "Введіть Email";

prompt = "Email:";

break;

case 3:

title = "Введіть адресу";

prompt = "Адреса:";

break;

case 4:

title = "Введіть логін";

prompt = "Логін:";

break;

}

// Передаємо дані у View через ViewBag

ViewBag.Title = title;

ViewBag.Prompt = prompt;

ViewBag.Error = TempData["Error"];

ViewBag.Success = TempData["Success"];

return View();

}

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Динамічний заголовок сторінки, який включає назву поточного етапу -->

<title>@ViewData["Title"] - Завдання 6 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN для швидкого старту -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключення власних CSS-стилів -->

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Навігаційна панель з назвою проекту -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Registration")">

Завдання 6 (Lab 5) – Реєстрація

</a>

</nav>

<!-- Основний контейнер для вмісту сторінки -->

<main class="container">

@RenderBody()

</main>

<!-- Футер з інформацією про авторські права -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Загальні стилі для повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використовуємо flexbox для організації елементів на сторінці \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає увесь доступний простір \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди залишається внизу \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних етапів реєстрації \*/

body.bg1 {

/\* Фон для етапу 1 (введення пароля) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-bhbe2xkopirx8goi.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для етапу 2 (введення Email) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-7h76hbdcrtpawe3i.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg3 {

/\* Фон для етапу 3 (введення адреси) \*/

background: url('https://wallpapers.com/images/hd/1920-x-1080-nature-desktop-3josvp4sieeo5ocn.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg4 {

/\* Фон для етапу 4 (введення логіну та підсумок) \*/

background: url('https://i.imgur.com/HaJuteg.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми: легкий прозорий фон з тінню \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація появи форми для плавності \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как дерево, вода, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, пейзаж, вода, облако

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как вода, текст, озеро, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, небо, текст, на открытом воздухе

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как дерево, небо, снимок экрана, гора

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит**:
   * Створення таблиці Accounts для зберігання даних користувачів у базі даних. Це включає поля для пароля, email, адреси, логіну, а також дату створення запису.
2. **Контролер RegistrationController**:
   * **Єдиний метод Index**:
     + **GET запит**: Перевіряється поточний етап реєстрації за допомогою значень у сесії. В залежності від того, яка інформація вже введена (пароль, email, адреса, логін), визначається поточний етап.
     + Встановлюється відповідний фон для сторінки, а також заголовок та підказка для поля вводу.
     + **POST запит**: Обробляється введена інформація, зберігається у сесії на відповідному етапі (пароль, email, адреса, логін).
     + Після введення всіх даних (логіну), вони записуються в базу даних.
     + Після успішного запису користувач отримує повідомлення про успіх, а сесія очищається для запобігання повторного сабміту.
3. **Інтерфейс (View)**:
   * Єдина форма, що динамічно змінює свій інтерфейс в залежності від поточного етапу.
   * Після кожного етапу введені дані зберігаються в сесії, і форма автоматично перенаправляє до наступного етапу.
   * Після завершення реєстрації відображається повідомлення про успіх і можливість розпочати нову реєстрацію.
4. **Запис у базу даних**:
   * Після введення всіх даних (пароль, email, адреса, логін), вони записуються в таблицю Accounts у базі даних.

## **Функціональність**

* **Поетапна реєстрація**: Користувач вводить дані поетапно, спочатку пароль, потім email, адресу і логін.
* **Динамічний інтерфейс**: Інтерфейс змінюється залежно від того, на якому етапі перебуває користувач (введення пароля, email, адреси, логіну).
* **Збереження в сесії**: Всі введені дані зберігаються в сесії між етапами для безперервного заповнення форми.
* **Запис у базу даних**: Після введення всіх даних (логіну) вони записуються в базу даних, що дозволяє зберігати інформацію про користувача.
* **Повідомлення про успіх або помилку**: Після завершення реєстрації або виникнення помилки відображається відповідне повідомлення.

## **Висновок**

Завдання демонструє ефективне використання сесій та єдиного методу контролера для обробки як GET, так і POST запитів. Реалізовано чотирьохетапну реєстрацію, де кожен етап зберігає введену інформацію та передає її до наступного етапу. Це дозволяє спростити структуру коду та уникнути дублювання логіки. Після завершення реєстрації всі дані записуються в базу даних. Система дозволяє користувачеві відчути інтуїтивно зрозумілий процес реєстрації з можливістю корекції помилок на кожному етапі.

# **Завдання 7**

Створити двох етапний WEB-процес пошуку абстрактного акаунту. На ко-жному етапі реалізується запит на інформацію згідно до власного варіанту. Після останнього етапу інформація має бути знайдена у відповідній таблиці бази даних та виведена на окремій View.

|  |  |  |
| --- | --- | --- |
| **№ варіанту** | **1 етап** | **2 етап** |
| 12 | Password | Email |

## **Текст програми:**

**SQL-запит:**

USE RegistrDB;

-- Створення таблиці Accounts для збереження даних акаунту (за варіантом №12)

CREATE TABLE Accounts (

Id INT IDENTITY(1,1) PRIMARY KEY,

Password NVARCHAR(100) NOT NULL,

Email NVARCHAR(100) NOT NULL,

CreatedAt DATETIME DEFAULT GETDATE()

);

**Password.cshtml:**

@{

// Використовуємо загальний макет сторінки

Layout = "~/Views/Shared/\_Layout.cshtml";

// Встановлюємо заголовок для цієї сторінки

ViewData["Title"] = "Введіть пароль для пошуку";

// Задаємо CSS-клас для фону (bg1)

ViewBag.BodyClass = "bg1";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок форми -->

<h2 class="text-center">Введіть пароль</h2>

@if (ViewBag.Error != null)

{

<!-- Якщо є помилка, відображаємо повідомлення -->

<div class="alert alert-danger">@ViewBag.Error</div>

}

<form asp-action="Password" method="post">

<div class="form-group">

<label for="password">Пароль:</label>

<input type="text" class="form-control" id="password" name="password" placeholder="Введіть пароль" required />

</div>

<!-- Кнопка для переходу до наступного кроку -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

</div>

</div>

</div>

**Email.cshtml:**

@{

// Підключаємо загальний макет

Layout = "~/Views/Shared/\_Layout.cshtml";

// Заголовок цієї сторінки

ViewData["Title"] = "Введіть Email для пошуку";

// Встановлюємо фон для цього етапу (bg2)

ViewBag.BodyClass = "bg2";

}

<div class="row justify-content-center">

<div class="col-md-6">

<div class="form-box">

<!-- Заголовок форми -->

<h2 class="text-center">Введіть Email</h2>

@if (ViewBag.Error != null)

{

<!-- Вивід помилки, якщо вона є -->

<div class="alert alert-danger">@ViewBag.Error</div>

}

<form asp-action="Email" method="post">

<div class="form-group">

<label for="email">Email:</label>

<input type="email" class="form-control" id="email" name="email" placeholder="Введіть Email" required />

</div>

<!-- Кнопка для запуску пошуку -->

<button type="submit" class="btn btn-primary btn-block">Пошук</button>

</form>

</div>

</div>

</div>

**Result.cshtml:**

@model Task\_7.Controllers.Account

@{

// Підключаємо загальний макет

Layout = "~/Views/Shared/\_Layout.cshtml";

// Заголовок сторінки результату

ViewData["Title"] = "Результат пошуку";

// Встановлюємо фон для сторінки результату

ViewBag.BodyClass = "bgResult";

}

<div class="row justify-content-center">

<div class="col-md-8">

<div class="form-box text-center">

<h2>Результат пошуку</h2>

@if (Model == null)

{

<!-- Якщо акаунт не знайдено, виводимо попередження -->

<div class="alert alert-warning">

Акаунт не знайдено.

</div>

}

else

{

<!-- Виводимо деталі знайденого акаунту у вигляді таблиці -->

<table class="table table-bordered">

<tr>

<th>ID</th>

<td>@Model.Id</td>

</tr>

<tr>

<th>Пароль</th>

<td>@Model.Password</td>

</tr>

<tr>

<th>Email</th>

<td>@Model.Email</td>

</tr>

<tr>

<th>Дата створення</th>

<td>@Model.CreatedAt</td>

</tr>

</table>

}

<!-- Кнопка для запуску нового пошукового запиту -->

<a href="@Url.Action("Password", "Search", new { reset = true })" class="btn btn-primary">Нова пошукова операція</a>

</div>

</div>

</div>

**SearchController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

using Newtonsoft.Json;

namespace Task\_7.Controllers

{

public class SearchController : Controller

{

private readonly IConfiguration \_configuration;

public SearchController(IConfiguration configuration)

{

// Зберігаємо конфігурацію для подальшого доступу до рядка підключення

\_configuration = configuration;

}

// View для введення Паролю

[HttpGet]

public IActionResult Password(bool reset = false)

{

// Якщо користувач вирішив почати знову – очищаємо сесію та помилки

if (reset)

{

HttpContext.Session.Clear();

TempData["Error"] = null;

}

// Встановлюємо фон (CSS-клас) для цієї сторінки

ViewBag.BodyClass = "bg1";

// Заголовок для цієї View

ViewData["Title"] = "Введіть пароль для пошуку";

return View();

}

[HttpPost]

public IActionResult Password(string password)

{

// Перевірка – якщо пароль не введено, повертаємо форму з помилкою

if (string.IsNullOrWhiteSpace(password))

{

ModelState.AddModelError("", "Будь ласка, введіть пароль.");

return View();

}

// Зберігаємо введений пароль у сесії

HttpContext.Session.SetString("SearchPassword", password);

return RedirectToAction("Email");

}

// View для введення Email

[HttpGet]

public IActionResult Email()

{

// Якщо пароль не заданий – повертаємо користувача на попередню сторінку

if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchPassword")))

return RedirectToAction("Password");

// Встановлюємо фон для цієї сторінки (інший CSS-клас)

ViewBag.BodyClass = "bg2";

ViewData["Title"] = "Введіть Email для пошуку";

return View();

}

[HttpPost]

public IActionResult Email(string email)

{

// Перевірка введення Email

if (string.IsNullOrWhiteSpace(email))

{

ModelState.AddModelError("", "Будь ласка, введіть Email.");

return View();

}

// Зберігаємо Email у сесії

HttpContext.Session.SetString("SearchEmail", email);

// Отримуємо дані, що були введені раніше

string searchPassword = HttpContext.Session.GetString("SearchPassword");

string searchEmail = HttpContext.Session.GetString("SearchEmail");

Account account = null;

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Підключаємося до бази даних і виконуємо запит для пошуку акаунту

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "SELECT TOP 1 Id, Password, Email, CreatedAt FROM Accounts WHERE Password = @Password AND Email = @Email";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", searchPassword);

cmd.Parameters.AddWithValue("@Email", searchEmail);

using (var reader = cmd.ExecuteReader())

{

if (reader.Read())

{

// Заповнюємо об'єкт account даними з бази

account = new Account

{

Id = reader.GetInt32(reader.GetOrdinal("Id")),

Password = reader.GetString(reader.GetOrdinal("Password")),

Email = reader.GetString(reader.GetOrdinal("Email")),

CreatedAt = reader.GetDateTime(reader.GetOrdinal("CreatedAt"))

};

}

}

}

}

}

catch (Exception ex)

{

// Якщо сталася помилка при пошуку, зберігаємо її повідомлення і повертаємо користувача на сторінку введення пароля

TempData["Error"] = "Помилка пошуку: " + ex.Message;

return RedirectToAction("Password", new { reset = true });

}

// Очищаємо сесію після завершення пошукового процесу

HttpContext.Session.Clear();

// Зберігаємо результат пошуку у TempData у форматі JSON для подальшого відображення

TempData["AccountResult"] = JsonConvert.SerializeObject(account);

return RedirectToAction("Result");

}

// View для відображення результату пошуку

[HttpGet]

public IActionResult Result()

{

// Встановлюємо фон для сторінки результату

ViewBag.BodyClass = "bgResult";

ViewData["Title"] = "Результат пошуку";

string json = TempData["AccountResult"] as string;

Account account = null;

if (!string.IsNullOrEmpty(json))

{

// Десеріалізуємо JSON-рядок у об'єкт Account

account = JsonConvert.DeserializeObject<Account>(json);

}

return View(account);

}

}

// Модель, що відповідає структурі таблиці Accounts

public class Account

{

public int Id { get; set; }

public string Password { get; set; }

public string Email { get; set; }

public DateTime CreatedAt { get; set; }

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Заголовок сторінки, що формується динамічно -->

<title>@ViewData["Title"] - Завдання 7 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключення власного CSS -->

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Верхня навігаційна панель з брендом -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Password", "Search")">

Завдання 7 (Lab 5) – Пошук акаунту

</a>

</nav>

<!-- Основний контейнер для відображення вмісту -->

<main class="container">

@RenderBody()

</main>

<!-- Футер із інформацією про авторські права -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">&copy; 2025 Lab 5</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Загальні стилі для повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Організація розташування елементів за допомогою flexbox \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає весь доступний простір \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди залишається внизу \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для різних сторінок \*/

body.bg1 {

/\* Фон для сторінки введення пароля \*/

background: url('https://wallpapercave.com/wp/wp3819690.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для сторінки введення Email \*/

background: url('https://s1.1zoom.me/b5050/330/USA\_Mountains\_Forests\_469872\_1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bgResult {

/\* Фон для сторінки результату пошуку \*/

background: url('https://www.hdwallpapers.in/download/granite\_boulders\_joshua\_tree\_national\_park\_california\_usa-1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми: прозорий фон із заокругленими кутами і тінню \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как Мультимедийное программное обеспечение, снимок экрана, программное обеспечение, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как небо, снимок экрана, облако, текст

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, дерево, облако, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как снимок экрана, текст, небо, гора

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит**:
   * Створення таблиці Accounts в базі даних для зберігання даних акаунтів користувачів, включаючи поля для пароля, електронної пошти та дати створення акаунту.
2. **Контролер SearchController**:
   * **Метод Password (GET і POST)**:
     + **GET**: Відображає сторінку для введення пароля.
     + **POST**: Перевіряє введений пароль, зберігає його в сесії і перенаправляє на сторінку введення електронної пошти.
   * **Метод Email (GET і POST)**:
     + **GET**: Перевіряє, чи пароль уже введено. Якщо ні, перенаправляє на сторінку введення пароля.
     + **POST**: Перевіряє введену електронну пошту, зберігає її в сесії, після чого виконує запит до бази даних для пошуку акаунту за паролем та email. Результат пошуку передається на сторінку результату.
   * **Метод Result (GET)**:
     + Відображає результат пошуку, де виводиться інформація про знайдений акаунт (ID, пароль, email, дата створення). Якщо акаунт не знайдено, показує повідомлення про помилку.
3. **Інтерфейс (Views)**:
   * **Password.cshtml**: Форма для введення пароля.
   * **Email.cshtml**: Форма для введення електронної пошти.
   * **Result.cshtml**: Сторінка для відображення результату пошуку, де показується інформація про знайдений акаунт або повідомлення про відсутність акаунту.
4. **Процес пошуку**:
   * Користувач спочатку вводить пароль, потім електронну пошту. Дані зберігаються в сесії та використовуються для пошуку в базі даних.
   * Якщо акаунт знайдено, виводиться його інформація, якщо ні – виводиться повідомлення про помилку.

## **Функціональність**

* **Поетапний пошук акаунту**: Користувач по черзі вводить пароль і email для пошуку акаунту.
* **Динамічна зміна інтерфейсу**: Інтерфейс змінюється в залежності від етапу пошуку (введення пароля, введення email, результат пошуку).
* **Перевірка введених даних**: Для кожного етапу перевіряється, чи введено необхідне значення (пароль, email).
* **Пошук в базі даних**: Виконується запит до бази даних для пошуку акаунту за введеними даними (пароль, email).
* **Результат пошуку**: Якщо акаунт знайдений, виводиться його інформація; якщо ні – повідомлення про відсутність акаунту.

## **Висновок**

Завдання реалізує двоетапний процес пошуку акаунту через веб-інтерфейс, де користувач спочатку вводить пароль, а потім email для пошуку в базі даних. Всі введені дані зберігаються в сесії, що дозволяє користувачу легко переходити між етапами. Після завершення пошуку результат відображається на окремій сторінці. Завдання демонструє базову роботу з сесіями та базами даних у контексті пошукових операцій в веб-додатках.

# **\*Завдання 8**

Виконати минуле завдання за допомогою однієї View, яка змінює свій ін-терфейс залежності від кількості введених даних майбутнього акаунту та одного Controller, який зберігає чергову порцію інформації акаунту в зале-жності від введеної інформації та виводить знайдені акаунти на цій же View.

## **Текст програми:**

**Index.cshtml:**

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

// ViewBag.Stage: 1 – введення Password, 2 – введення Email, 3 – відображення результатів

int stage = ViewBag.Stage;

// Заголовок, який показуємо на формі (наприклад, "Введіть пароль для пошуку")

string title = ViewBag.Title;

// Підказка для поля вводу, яка змінюється залежно від етапу

string prompt = ViewBag.Prompt;

// Повідомлення про помилку, якщо вона виникла під час введення

string error = ViewBag.Error as string;

// Для етапу 3 результати передаються через ViewBag.SearchResults як List<Account>

var searchResults = ViewBag.SearchResults as List<Task\_8.Controllers.Account>;

}

<div class="row justify-content-center">

<div class="col-md-8">

<div class="form-box">

<!-- Вивід заголовку форми -->

<h2 class="text-center">@title</h2>

@if (!string.IsNullOrEmpty(error))

{

<!-- Якщо виникла помилка, виводимо її в червоному блоці -->

<div class="alert alert-danger">@error</div>

}

@if (stage == 3)

{

<!-- Якщо ми на етапі відображення результатів -->

@if (searchResults == null || searchResults.Count == 0)

{

<!-- Якщо акаунти не знайдено, виводимо попередження -->

<div class="alert alert-warning">Акаунт не знайдено.</div>

}

else

{

<!-- Якщо результати є, виводимо їх у таблиці -->

<table class="table table-bordered">

<tr>

<th>ID</th>

<th>Пароль</th>

<th>Email</th>

<th>Дата створення</th>

</tr>

@foreach (var acc in searchResults)

{

<tr>

<td>@acc.Id</td>

<td>@acc.Password</td>

<td>@acc.Email</td>

<td>@acc.CreatedAt</td>

</tr>

}

</table>

}

<!-- Кнопка для початку нового пошукового запиту -->

<a href="@Url.Action("Index", "Search", new { reset = true })" class="btn btn-primary">Нова пошукова операція</a>

}

else

{

<!-- Якщо це не етап відображення результатів – виводимо форму вводу -->

<form asp-action="Index" method="post">

<div class="form-group">

<label for="input">@prompt</label>

<input type="text" id="input" name="input" class="form-control" placeholder="@prompt" required />

</div>

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

}

</div>

</div>

</div>

**SearchController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

using Newtonsoft.Json;

namespace Task\_8.Controllers

{

public class SearchController : Controller

{

private readonly IConfiguration \_configuration;

public SearchController(IConfiguration configuration)

{

// Зберігаємо конфігурацію для роботи з базою даних

\_configuration = configuration;

}

// GET: Search/Index

[HttpGet]

public IActionResult Index(bool reset = false)

{

// Якщо користувач хоче розпочати новий пошук – очищуємо сесію та TempData

if (reset)

{

HttpContext.Session.Clear();

TempData["SearchResults"] = null;

return RedirectToAction("Index");

}

// Якщо у TempData є результати пошуку, переходимо до етапу 3

var tempResults = TempData["SearchResults"] as string;

if (!string.IsNullOrEmpty(tempResults))

{

ViewBag.Stage = 3;

// Десеріалізуємо результати з JSON-рядка у список акаунтів

var results = JsonConvert.DeserializeObject<List<Account>>(tempResults);

ViewBag.SearchResults = results;

// Встановлюємо фон для результату

ViewBag.BodyClass = "bg2";

ViewBag.Title = "Результати пошуку";

ViewBag.Prompt = "";

return View();

}

// Якщо результатів немає, визначаємо етап за даними сесії

int stage = 1;

if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchPassword")))

stage = 1;

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchEmail")))

stage = 2;

ViewBag.Stage = stage;

// Встановлюємо CSS-клас для фону залежно від етапу

if (stage == 1)

ViewBag.BodyClass = "bg1";

else

ViewBag.BodyClass = "bg2";

// Встановлюємо заголовок та підказку для форми залежно від етапу

if (stage == 1)

{

ViewBag.Title = "Введіть пароль для пошуку";

ViewBag.Prompt = "Пароль:";

}

else if (stage == 2)

{

ViewBag.Title = "Введіть Email для пошуку";

ViewBag.Prompt = "Email:";

}

ViewBag.Error = TempData["Error"];

return View();

}

// POST: Search/Index

[HttpPost]

public IActionResult Index(string input)

{

// Етап 1: Якщо ще не введено пароль, зберігаємо введене значення як пароль

if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchPassword")))

{

HttpContext.Session.SetString("SearchPassword", input);

return RedirectToAction("Index");

}

// Етап 2: Якщо пароль введено, а Email ще не збережено – зберігаємо Email та виконуємо пошук

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchEmail")))

{

HttpContext.Session.SetString("SearchEmail", input);

string searchPassword = HttpContext.Session.GetString("SearchPassword");

string searchEmail = HttpContext.Session.GetString("SearchEmail");

List<Account> accounts = new List<Account>();

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Підключаємося до бази та виконуємо запит для пошуку акаунтів

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "SELECT Id, Password, Email, CreatedAt FROM Accounts WHERE Password = @Password AND Email = @Email";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", searchPassword);

cmd.Parameters.AddWithValue("@Email", searchEmail);

using (SqlDataReader reader = cmd.ExecuteReader())

{

// Збираємо всі знайдені записи в список

while (reader.Read())

{

accounts.Add(new Account

{

Id = reader.GetInt32(reader.GetOrdinal("Id")),

Password = reader.GetString(reader.GetOrdinal("Password")),

Email = reader.GetString(reader.GetOrdinal("Email")),

CreatedAt = reader.GetDateTime(reader.GetOrdinal("CreatedAt"))

});

}

}

}

}

}

catch (Exception ex)

{

// Якщо виникла помилка під час пошуку, зберігаємо повідомлення і повертаємося на початок

TempData["Error"] = "Помилка пошуку: " + ex.Message;

return RedirectToAction("Index", new { reset = true });

}

// Очищаємо сесію після завершення пошуку, щоб уникнути повторного сабміту

HttpContext.Session.Clear();

// Зберігаємо результати пошуку у TempData у форматі JSON для подальшого відображення

TempData["SearchResults"] = JsonConvert.SerializeObject(accounts);

return RedirectToAction("Index");

}

return RedirectToAction("Index");

}

}

// Клас моделі, що відповідає структурі таблиці Accounts

public class Account

{

public int Id { get; set; }

public string Password { get; set; }

public string Email { get; set; }

public DateTime CreatedAt { get; set; }

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Динамічний заголовок, що формується на основі ViewData -->

<title>@ViewData["Title"] - Завдання 8 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap через CDN для швидкого налаштування зовнішнього вигляду -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключення власних стилів -->

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Верхня навігаційна панель з брендом -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Search")">

Завдання 8 (Lab 5) – Пошук акаунту

</a>

</nav>

<!-- Основний контейнер для відображення вмісту сторінки -->

<main class="container">

@RenderBody()

</main>

<!-- Футер з інформацією про авторські права -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Забезпечення повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використовуємо flexbox для організації контенту \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає всю доступну висоту \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди розташовується внизу \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для етапів пошуку \*/

body.bg1 {

/\* Фон для сторінки введення пароля \*/

background: url('https://wallpapercave.com/wp/wp3819690.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для сторінки введення Email або результатів пошуку \*/

background: url('https://s1.1zoom.me/b5050/330/USA\_Mountains\_Forests\_469872\_1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bgResult {

/\* Фон для сторінки з результатами пошуку \*/

background: url('https://www.hdwallpapers.in/download/granite\_boulders\_joshua\_tree\_national\_park\_california\_usa-1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми: легкий прозорий фон, заокруглені кути та тінь \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація плавного появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как небо, текст, облако, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, облако, дерево, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, облако, дерево, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит для створення таблиці Accounts**:
   * Створюється таблиця Accounts у базі даних для зберігання акаунтів користувачів з полями для пароля, email і дати створення.
2. **Контролер SearchController**:
   * **GET метод Index**:
     + Перевіряється поточний етап пошуку, що зберігається в сесії, щоб визначити, чи користувач знаходиться на етапі введення пароля, email чи перегляду результатів.
     + Якщо є результати в TempData, то вони десеріалізуються і відображаються на сторінці.
     + Для кожного етапу встановлюється відповідний CSS-клас фону і відображається форма або результати пошуку.
   * **POST метод Index**:
     + Якщо на етапі 1 (пароль), введений пароль зберігається в сесії, і користувач переходить до етапу введення email.
     + Якщо на етапі 2 (email), введений email зберігається в сесії, після чого виконується запит до бази даних для пошуку акаунту за паролем і email.
     + Результати пошуку передаються на ту ж сторінку і відображаються в таблиці.
3. **Інтерфейс (Views)**:
   * **Index.cshtml**:
     + Форма для введення пароля на першому етапі.
     + Форма для введення email на другому етапі.
     + Якщо результати пошуку є, вони відображаються у вигляді таблиці.
     + Повідомлення про успіх або помилку також відображаються на тій же сторінці.
4. **Робота з результатами**:
   * Після введення пароля і email дані зберігаються в сесії, а потім використовується запит до бази даних для пошуку акаунтів, які відповідають введеним критеріям.
   * Якщо акаунти знайдено, вони відображаються на сторінці результатів. Якщо акаунти не знайдено, виводиться повідомлення про це.

## **Функціональність**

* **Поетапний процес пошуку акаунтів**: Користувач по черзі вводить пароль та email для пошуку акаунтів.
* **Динамічна зміна інтерфейсу**: В залежності від поточного етапу, інтерфейс змінюється, відображаючи або форму для введення даних, або таблицю з результатами пошуку.
* **Збереження даних у сесії**: Пароль і email зберігаються в сесії, щоб забезпечити перехід між етапами пошуку.
* **Запит до бази даних**: Після введення пароля та email виконується запит до бази даних для пошуку акаунтів.
* **Результати пошуку**: Якщо акаунти знайдено, вони відображаються в таблиці, якщо ні — виводиться повідомлення про відсутність акаунтів.

## **Висновок**

Завдання дозволяє створити інтерактивний двоетапний процес пошуку акаунтів за паролем і email, що забезпечує гнучкість і зручність користувача. За допомогою однієї сторінки і одного контролера ми змогли реалізувати динамічне відображення форм та результатів пошуку, а також обробку введених даних за допомогою сесій. Це дозволяє значно спростити структуру програми і полегшити підтримку, зберігаючи при цьому всі необхідні функції для виконання пошуку в базі даних.

# **\*Завдання 9**

Виконати минуле завдання, використовуючи лише один метод в контро-лері.

## **Текст програми:**

**Index.cshtml:**

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

// Безпечне приведення ViewBag.Stage до int із значенням за замовчуванням 1

int stage = ViewBag.Stage != null ? (int)ViewBag.Stage : 1;

// Заголовок форми. Якщо нічого не задано – виводимо стандартний текст

string title = ViewBag.Title ?? "Введіть дані для пошуку";

// Підказка для поля вводу (наприклад, "Пароль:" або "Email:")

string prompt = ViewBag.Prompt ?? "";

// Отримуємо повідомлення про помилку, якщо вона є

string error = ViewBag.Error as string;

// Для етапу 3 результати пошуку передаються через ViewBag.SearchResults як List<Account>

var searchResults = ViewBag.SearchResults as List<Task\_9.Controllers.Account>;

}

<div class="row justify-content-center">

<div class="col-md-8">

<div class="form-box">

<!-- Виводимо заголовок форми -->

<h2 class="text-center">@title</h2>

@if (!string.IsNullOrEmpty(error))

{

<!-- Якщо виникла помилка, показуємо її користувачу -->

<div class="alert alert-danger">@error</div>

}

@if (stage == 3)

{

<!-- Етап 3: відображення результатів пошуку -->

@if (searchResults == null || searchResults.Count == 0)

{

<!-- Якщо пошук не дав результатів, виводимо попередження -->

<div class="alert alert-warning">Акаунт не знайдено.</div>

}

else

{

<!-- Вивід результатів пошуку у вигляді таблиці -->

<table class="table table-bordered">

<tr>

<th>ID</th>

<th>Пароль</th>

<th>Email</th>

<th>Дата створення</th>

</tr>

@foreach (var acc in searchResults)

{

<tr>

<td>@acc.Id</td>

<td>@acc.Password</td>

<td>@acc.Email</td>

<td>@acc.CreatedAt</td>

</tr>

}

</table>

}

<!-- Кнопка для запуску нового пошукового запиту -->

<a href="@Url.Action("Index", "Search", new { reset = true })" class="btn btn-primary">Нова пошукова операція</a>

}

else

{

<!-- Якщо не етап відображення результатів – показуємо форму вводу -->

<form asp-action="Index" method="post">

<div class="form-group">

<label for="input">@prompt</label>

<input type="text" id="input" name="input" class="form-control" placeholder="@prompt" required />

</div>

<!-- Кнопка для відправлення форми -->

<button type="submit" class="btn btn-primary btn-block">Далі</button>

</form>

}

</div>

</div>

</div>

**SearchController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System;

using System.Collections.Generic;

using Microsoft.Extensions.Configuration;

using Microsoft.Data.SqlClient;

using Newtonsoft.Json;

namespace Task\_9.Controllers

{

public class SearchController : Controller

{

private readonly IConfiguration \_configuration;

public SearchController(IConfiguration configuration)

{

// Зберігаємо конфігурацію для доступу до рядка підключення до БД

\_configuration = configuration;

}

// Єдиний метод, що обробляє як GET, так і POST запити

public IActionResult Index(string input = null, bool reset = false)

{

// Якщо користувач хоче почати новий пошук, очищуємо сесію і TempData

if (reset)

{

HttpContext.Session.Clear();

TempData["SearchResults"] = null;

return RedirectToAction("Index");

}

// Якщо запит POST та значення введено – обробляємо дані

if (Request.Method.Equals("POST", StringComparison.OrdinalIgnoreCase) && !string.IsNullOrEmpty(input))

{

// Етап 1: Якщо в сесії ще не збережено "SearchPassword", зберігаємо введене значення як пароль

if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchPassword")))

{

HttpContext.Session.SetString("SearchPassword", input);

return RedirectToAction("Index");

}

// Етап 2: Якщо пароль введено, а "SearchEmail" ще не збережено – зберігаємо email і виконуємо пошук

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchEmail")))

{

HttpContext.Session.SetString("SearchEmail", input);

string searchPassword = HttpContext.Session.GetString("SearchPassword");

string searchEmail = HttpContext.Session.GetString("SearchEmail");

List<Account> accounts = new List<Account>();

string connStr = \_configuration.GetConnectionString("RegistrationDb");

try

{

// Підключаємось до бази і виконуємо SQL-запит

using (SqlConnection conn = new SqlConnection(connStr))

{

conn.Open();

string sql = "SELECT Id, Password, Email, CreatedAt FROM Accounts WHERE Password = @Password AND Email = @Email";

using (SqlCommand cmd = new SqlCommand(sql, conn))

{

cmd.Parameters.AddWithValue("@Password", searchPassword);

cmd.Parameters.AddWithValue("@Email", searchEmail);

using (SqlDataReader reader = cmd.ExecuteReader())

{

// Збираємо всі знайдені записи

while (reader.Read())

{

accounts.Add(new Account

{

Id = reader.GetInt32(reader.GetOrdinal("Id")),

Password = reader.GetString(reader.GetOrdinal("Password")),

Email = reader.GetString(reader.GetOrdinal("Email")),

CreatedAt = reader.GetDateTime(reader.GetOrdinal("CreatedAt"))

});

}

}

}

}

}

catch (Exception ex)

{

// Якщо виникла помилка, зберігаємо повідомлення та повертаємося до першого етапу

TempData["Error"] = "Помилка пошуку: " + ex.Message;

return RedirectToAction("Index", new { reset = true });

}

// Очищуємо сесію, щоб не повертатися до етапу вводу даних

HttpContext.Session.Clear();

// Зберігаємо результати пошуку у TempData у форматі JSON

TempData["SearchResults"] = JsonConvert.SerializeObject(accounts);

// Перехід до GET-логіки, яка встановить stage = 3 і відобразить результати

return RedirectToAction("Index");

}

}

// Обробка GET-запиту:

// Якщо у TempData є результати пошуку – встановлюємо stage = 3

string tempJson = TempData["SearchResults"] as string;

if (!string.IsNullOrEmpty(tempJson))

{

ViewBag.Stage = 3;

var results = JsonConvert.DeserializeObject<List<Account>>(tempJson);

ViewBag.SearchResults = results;

ViewBag.BodyClass = "bg2";

ViewBag.Title = "Результати пошуку";

ViewBag.Prompt = "";

return View();

}

// Якщо результатів немає, визначаємо stage за сесійними даними

int stageVal = 1;

if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchPassword")))

{

stageVal = 1;

}

else if (string.IsNullOrEmpty(HttpContext.Session.GetString("SearchEmail")))

{

stageVal = 2;

}

ViewBag.Stage = stageVal;

// Встановлюємо фон, заголовок і підказку залежно від stage

if (stageVal == 1)

{

ViewBag.BodyClass = "bg1";

ViewBag.Title = "Введіть пароль для пошуку";

ViewBag.Prompt = "Пароль:";

}

else if (stageVal == 2)

{

ViewBag.BodyClass = "bg2";

ViewBag.Title = "Введіть Email для пошуку";

ViewBag.Prompt = "Email:";

}

// Передаємо повідомлення про помилку (якщо є)

ViewBag.Error = TempData["Error"];

return View();

}

}

// Клас моделі, що відповідає структурі таблиці Accounts

public class Account

{

public int Id { get; set; }

public string Password { get; set; }

public string Email { get; set; }

public DateTime CreatedAt { get; set; }

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<!-- Заголовок сторінки формується динамічно через ViewData -->

<title>@ViewData["Title"] - Завдання 9 (Lab 5)</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<!-- Підключення Bootstrap для швидкого налаштування зовнішнього вигляду -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" />

<!-- Підключення власних стилів -->

<link rel="stylesheet" type="text/css" href="~/css/site.css" />

</head>

<body class="@ViewBag.BodyClass">

<!-- Верхня навігаційна панель з логотипом та назвою -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="@Url.Action("Index", "Search")">

Завдання 9 (Lab 5) – Пошук акаунту

</a>

</nav>

<!-- Основний контейнер для відображення вмісту сторінки -->

<main class="container">

@RenderBody()

</main>

<!-- Футер із копірайтами -->

<footer class="footer bg-dark text-white text-center p-3">

<div class="container">

&copy; 2025 Lab 5

</div>

</footer>

<!-- Підключення JavaScript для Bootstrap -->

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

**site.css:**

/\* Забезпечення повноекранного відображення \*/

html, body {

height: 100%;

margin: 0;

padding: 0;

}

/\* Використання flexbox для розташування вмісту \*/

body {

display: flex;

flex-direction: column;

font-family: Arial, sans-serif;

}

/\* Основний контейнер займає увесь доступний простір \*/

main {

flex: 1 0 auto;

}

/\* Футер завжди залишається внизу сторінки \*/

footer {

flex-shrink: 0;

}

/\* Фонові зображення для етапів пошуку \*/

body.bg1 {

/\* Фон для сторінки введення пароля \*/

background: url('https://wallpapercave.com/wp/wp3819690.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bg2 {

/\* Фон для сторінки введення email або результатів пошуку \*/

background: url('https://s1.1zoom.me/b5050/330/USA\_Mountains\_Forests\_469872\_1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

body.bgResult {

/\* Фон для сторінки з результатами пошуку \*/

background: url('https://www.hdwallpapers.in/download/granite\_boulders\_joshua\_tree\_national\_park\_california\_usa-1920x1080.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* Стилізація блоку форми: прозорий фон із заокругленими кутами та тінню \*/

.form-box {

background-color: rgba(255, 255, 255, 0.9);

padding: 30px;

border-radius: 10px;

box-shadow: 0 4px 6px rgba(0,0,0,0.1);

animation: fadeIn 1s ease-out;

margin-top: 30px;

}

/\* Анімація плавної появи форми \*/

@keyframes fadeIn {

from {

opacity: 0;

transform: translateY(-20px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

## **Результати роботи програми:**

Изображение выглядит как Мультимедийное программное обеспечение, программное обеспечение, текст, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как небо, текст, облако, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, дерево, облако, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, облако, дерево, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

## **Алгоритм побудови кодів**

1. **SQL-запит**:
   * Створюється таблиця Accounts в базі даних, щоб зберігати акаунти користувачів з полями для пароля, email і дати створення.
2. **Контролер SearchController**:
   * **Єдиний метод Index**:
     + **GET запит**:
       - Якщо користувач хоче розпочати новий пошук, очищуємо сесію і TempData.
       - Якщо є результати пошуку в TempData, десеріалізуємо їх і відображаємо на сторінці.
       - Якщо результатів немає, перевіряється поточний етап (пароль чи email), і виводиться відповідна форма.
       - Встановлюється відповідний CSS-клас фону і заголовок для кожного етапу.
     + **POST запит**:
       - На першому етапі зберігається введений пароль в сесію.
       - На другому етапі, якщо пароль введений, зберігається email, після чого виконується пошук у базі даних.
       - Пошук акаунтів виконується за введеними паролем та email. Результати зберігаються в TempData у форматі JSON.
3. **Інтерфейс (Views)**:
   * **Index.cshtml**:
     + Форма для введення пароля на першому етапі.
     + Форма для введення email на другому етапі.
     + Якщо результати пошуку є, вони відображаються у вигляді таблиці.
     + Якщо акаунти не знайдено, виводиться повідомлення про відсутність акаунтів.
     + Якщо сталася помилка, показується повідомлення про помилку.
4. **Робота з результатами**:
   * Після введення пароля та email виконується запит до бази даних для пошуку акаунтів, які відповідають введеним критеріям.
   * Якщо акаунти знайдено, вони відображаються на сторінці результатів. Якщо акаунти не знайдено, виводиться повідомлення про це.

## **Функціональність**

* **Поетапний процес пошуку акаунтів**: Користувач по черзі вводить пароль та email для пошуку акаунтів.
* **Динамічна зміна інтерфейсу**: В залежності від поточного етапу, інтерфейс змінюється, відображаючи або форму для введення даних, або таблицю з результатами пошуку.
* **Збереження даних у сесії**: Пароль і email зберігаються в сесії, що дозволяє зберігати стан між етапами.
* **Запит до бази даних**: Після введення пароля і email виконується запит до бази даних для пошуку акаунтів.
* **Результати пошуку**: Якщо акаунти знайдено, вони відображаються в таблиці. Якщо акаунти не знайдено, виводиться відповідне повідомлення.

## **Висновок**

Завдання реалізує двоетапний процес пошуку акаунтів за паролем і email через одну сторінку та один контролер. Це дозволяє оптимізувати структуру додатку, зменшуючи кількість необхідних методів в контролері, і забезпечує зручний інтерфейс для користувача. Всі введені дані зберігаються в сесії, що дозволяє плавно переходити між етапами. Результати пошуку відображаються на тій самій сторінці, що робить додаток більш зручним і інтерактивним.

# **Загальний Висновок**

Лабораторна робота №5 демонструє практичне застосування технологій ASP.NET для розробки серверних застосунків, що включає реалізацію різних підходів до передачі даних між контролерами та поданнями, перевірки коректності введених даних і обчислення параметрів (наприклад, перевірку наявності тупого кута у трикутнику). В роботі було реалізовано кілька варіантів вирішення завдання:

* **Мультиконтролерний підхід** – дані вводяться через окремі сторінки, де використовується TempData та сесія для збереження інформації, а логіка розрахунку розміщена у відповідному контролері.
* **Інтерфейс на одній View** – завдяки використанню єдиного контролера з динамічним інтерфейсом користувача, що змінюється в залежності від кількості введених сторін, зменшується дублювання коду і покращується зручність використання.
* **Реалізація з одним методом** – спрощений варіант, де один метод обробляє як GET, так і POST запити, що дозволяє ефективніше управляти логікою вводу та обчислення.

Крім того, завдання з реєстрації абстрактного акаунту демонструє побудову чотирьохетапного процесу, де на кожному кроці здійснюється перевірка коректності введених даних, а кінцеві результати записуються до бази даних. Це сприяє засвоєнню принципів роботи із сесіями, обробки форм та інтеграції з SQL-сервером.

Загалом, лабораторна робота сприяє закріпленню базових знань з проектування серверних застосунків, дозволяючи студенту освоїти як розбивати складні процеси на логічно організовані етапи, так і ефективно використовувати можливості MVC-архітектури для побудови інтерактивних та надійних веб-додатків. Це є важливим кроком у формуванні практичних навичок розробки сучасних інформаційних систем.