Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Інформатики та інтелектуальної власності"

**Звіт**

**Лабораторна робота 5**

з дисципліни "Проєктування серверних застосунків"

Виконав: студент групи КН-922В

Шершньов М.О.

Перевірив:

Івашко А.В.

Харків 2025

Зміст

[Завдання 3](#_Toc194086659)

[Класи maxtriangleside, accountregistration, accountsearch, singleclass, singlemethod 5](#_Toc194086660)

[Клас MaxTriangleSide 5](#_Toc194086661)

[Клас AccountRegistration 7](#_Toc194086662)

[Клас AccountSearch 10](#_Toc194086663)

[Клас SingleClass 13](#_Toc194086664)

[Клас SingleMethod 16](#_Toc194086665)

[Висновок 19](#_Toc194086666)

**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

Завдання

1. Створити зв'язку Controller1 - View1 - 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;

Мій варіант:





2. \*Виконати минуле завдання за допомогою однієї View яка змінює інтерфейс 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;

3. Виконати минуле завдання, використовуючи лише один метод в контролері;

4. Створити чотирьох етапний WEB-процес реєстрації абстрактного акаунту. На кожному етапі реалізується запит на інформацію згідно до власного варіанту. Після останнього етапу інформація має бути записана до відповідної таблиці бази даних;

Мій варіант:





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;

6. \*Виконати минуле завдання, використовуючи лише один метод в контролері;

7. Створити двох етапний WEВ-процес пошуку абстрактного акаунту. На кожному 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;

Мій варіант:





8. \*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 виводить знайдені акаунти на цій же View.

9. \*Виконати минуле завдання, використовуючи лише один метод в контролері.

Класи для роботи:

1. «Максимальна сторона трикутника» - **MaxTriangleSide**
2. «Реєстрація акаунту» – **AccountRegistration**
3. «Пошук акаунту» – **AccountSearch**
4. «Один клас» – **SingleClass**
5. «Один метод» – **SingleMethod**

Класи maxtriangleside, accountregistration, accountsearch, singleclass, singlemethod

Клас MaxTriangleSide

Клас **MaxTriangleSide** визначає найбільшу сторону трикутника, яку користувач вводить поетапно через три різні контролери (Side1Controller, Side2Controller, Side3Controller).

Основні функції класу:

**1. Отримує значення сторін трикутника** через TempData, які передаються з інших контролерів.

**2. Конвертує їх у** double для подальших обчислень.

**3. Знаходить максимальне значення серед трьох сторін** за допомогою Max().

**4. Передає результат у** ViewBag.MaxSide для відображення на сторінці.

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Side1Controller : Controller

{

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult SaveSide(double side1)

{

TempData["Side1"] = side1.ToString();

return RedirectToAction("Index", "Side2");

}

}

}

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Side2Controller : Controller

{

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult SaveSide(double side2)

{

TempData["Side2"] = side2.ToString();

return RedirectToAction("Index", "Side3");

}

}

}

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class Side3Controller : Controller

{

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult SaveSide(double side3)

{

TempData["Side3"] = side3.ToString();

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

}

}

}

using Microsoft.AspNetCore.Mvc;

namespace Task\_1.Controllers

{

public class TriangleController : Controller

{

public IActionResult Index()

{

double side1 = Convert.ToDouble(TempData["Side1"] ?? "0");

double side2 = Convert.ToDouble(TempData["Side2"] ?? "0");

double side3 = Convert.ToDouble(TempData["Side3"] ?? "0");

ViewBag.MaxSide = new double[] { side1, side2, side3 }.Max();

return View();

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 5.1-5.4 – Результат роботи класу MaxTriangleSide

Клас AccountRegistration

Клас **AccountRegistration** відповідає за покрокову реєстрацію користувача.

Основні функції:

– **Отримання даних користувача** – поетапне введення адреси, пароля, логіна та електронної пошти через відповідні контролери.

– **Валідація** – перевірка, щоб кожне поле не було порожнім (якщо так – повертається помилка).

– **Передача даних через TempData** – збереження проміжних значень між запитами.

– **Збереження у базі даних** – після введення всіх даних (EmailController) створюється об'єкт User, який додається до БД.

using Microsoft.AspNetCore.Mvc;

namespace Task\_4.Controllers

{

public class AddressController : Controller

{

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult Index(string address)

{

if (string.IsNullOrEmpty(address))

{

ViewData["Error"] = "Адреса не може бути порожньою.";

return View();

}

TempData["Address"] = address;

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

}

}

}

using Microsoft.AspNetCore.Mvc;

namespace Task\_4.Controllers

{

public class PasswordController : Controller

{

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult Index(string password)

{

if (string.IsNullOrEmpty(password))

{

ViewData["Error"] = "Пароль не може бути порожнім.";

return View();

}

TempData["Password"] = password;

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

}

}

}

using Microsoft.AspNetCore.Mvc;

namespace Task\_4.Controllers

{

public class LoginController : Controller

{

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult Index(string login)

{

if (string.IsNullOrEmpty(login))

{

ViewData["Error"] = "Логін не може бути порожнім.";

return View();

}

TempData["Login"] = login;

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

}

}

}

using Microsoft.AspNetCore.Mvc;

using Task\_4.Data;

using Task\_4.Models;

namespace Task\_4.Controllers

{

public class EmailController : Controller

{

private readonly ApplicationDbContext \_context;

public EmailController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult Index(string email)

{

if (string.IsNullOrEmpty(email))

{

ViewData["Error"] = "Електронна пошта не може бути порожньою.";

return View();

}

var address = TempData["Address"] as string;

var password = TempData["Password"] as string;

var login = TempData["Login"] as string;

if (string.IsNullOrEmpty(address) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(login))

{

ViewData["Error"] = "Помилка при отриманні даних. Будь ласка, почніть спочатку.";

return View();

}

var user = new User

{

Address = address,

Password = password,

Login = login,

Email = email

};

\_context.Users.Add(user);

\_context.SaveChanges();

return RedirectToAction("Success");

}

public IActionResult Success()

{

return View();

}

}

}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 5.5-5.9 – Результат роботи класу AccountRegistration

Клас AccountSearch

Клас **AccountSearch** відповідає за пошук облікового запису в базі даних за адресою та паролем.

**Основні функції:**

– **Пошук користувача за адресою** (AddressController) – перевіряє, чи існує користувач із введеною адресою. Якщо ні – виводить помилку.

– **Перехід до перевірки пароля** – якщо користувач знайдений, збереження адреси в TempData і перенаправлення до PasswordController.

– **Перевірка пароля** (PasswordController) – шукає користувача з відповідною адресою та паролем. Якщо введений пароль невірний – виводиться помилка.

– **Відображення результату** – при успішному вході дані користувача передаються у Result для відображення.

using Microsoft.AspNetCore.Mvc;

using Task\_7.Data;

namespace Task\_7.Controllers

{

public class AddressController : Controller

{

private readonly AppDbContext \_context;

public AddressController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

return View();

}

[HttpPost]

public IActionResult Index(string address)

{

var user = \_context.Users.FirstOrDefault(u => u.Address == address);

if (user == null)

{

ViewData["Error"] = "Користувача не знайдено за вказаною адресою.";

return View();

}

TempData["Address"] = address;

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

}

}

}

using Microsoft.AspNetCore.Mvc;

using Task\_7.Data;

namespace Task\_7.Controllers

{

public class PasswordController : Controller

{

private readonly AppDbContext \_context;

public PasswordController(AppDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index()

{

if (!TempData.ContainsKey("Address"))

{

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

}

return View();

}

[HttpPost]

public IActionResult Index(string password)

{

string? address = TempData["Address"] as string;

if (address == null) return RedirectToAction("Index", "Address");

var user = \_context.Users.FirstOrDefault(u => u.Address == address && u.Password == password);

if (user == null)

{

ViewData["Error"] = "Невірний пароль.";

return View();

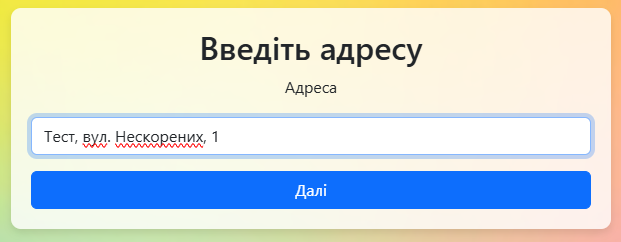
}

return View("Result", user);

}

}

}



Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунки 5.10-5.12 – Результат роботи класу AccountSearch

Клас SingleClass

Клас **SingleClass** у контексті вашого коду можна розглядати як узагальнену одиничну сутність, яка виконує всі необхідні операції в межах одного контролера.

using Microsoft.AspNetCore.Mvc;

using Task\_2.Extensions;

namespace Task\_2.Controllers

{

public class HomeController : Controller

{

private const string SessionKey = "TriangleSides";

public IActionResult Index()

{

List<double> sides = HttpContext.Session.GetObjectFromJson<List<double>>(SessionKey) ?? new List<double>();

ViewBag.Sides = sides;

ViewBag.MaxSide = sides.Count == 3 ? sides.Max() : (double?)null;

return View();

}

[HttpPost]

public IActionResult AddSide(double side)

{

if (side <= 0)

{

ModelState.AddModelError("", "Сторона повинна бути більше 0.");

return RedirectToAction("Index");

}

List<double> sides = HttpContext.Session.GetObjectFromJson<List<double>>(SessionKey) ?? new List<double>();

if (sides.Count < 3)

{

sides.Add(side);

HttpContext.Session.SetObjectAsJson(SessionKey, sides);

}

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult Reset()

{

HttpContext.Session.Remove(SessionKey);

return RedirectToAction("Index");

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Task\_5.Data;

using Task\_5.Models;

namespace Task\_5.Controllers

{

public class HomeController : Controller

{

private readonly ApplicationDbContext \_context;

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

[HttpGet]

public IActionResult Index(int step = 1)

{

ViewData["Step"] = step;

return View();

}

[HttpPost]

public IActionResult Index(int step, string input)

{

if (string.IsNullOrEmpty(input))

{

ViewData["Error"] = "Це поле не може бути порожнім.";

ViewData["Step"] = step;

return View();

}

switch (step)

{

case 1:

TempData["Address"] = input;

return RedirectToAction("Index", new { step = 2 });

case 2:

TempData["Password"] = input;

return RedirectToAction("Index", new { step = 3 });

case 3:

TempData["Login"] = input;

return RedirectToAction("Index", new { step = 4 });

case 4:

TempData["Email"] = input;

var user = new User

{

Address = TempData["Address"] as string ?? string.Empty,

Password = TempData["Password"] as string ?? string.Empty,

Login = TempData["Login"] as string ?? string.Empty,

Email = TempData["Email"] as string ?? string.Empty

};

try

{

\_context.Users.Add(user);

\_context.SaveChanges();

return RedirectToAction("Success");

}

catch (DbUpdateException ex)

{

var innerException = ex.InnerException?.Message ?? "No inner exception";

ViewData["Error"] = $"Не вдалося зберегти дані користувача: {innerException}";

ViewData["Step"] = step;

return View();

}

}

return View();

}

public IActionResult Success()

{

return View();

}

}

}

using Microsoft.AspNetCore.Mvc;

using Task\_8.Data;

namespace Task\_8.Controllers

{

public class HomeController : Controller

{

private readonly AppDbContext \_context;

public HomeController(AppDbContext context)

{

\_context = context;

}

public IActionResult Index()

{

ViewData["Step"] = HttpContext.Session.GetInt32("Step") ?? 1;

return View();

}

[HttpPost]

public IActionResult Index(string address, string password)

{

int step = HttpContext.Session.GetInt32("Step") ?? 1;

if (step == 1)

{

if (string.IsNullOrEmpty(address))

{

ViewData["Error"] = "Введіть адресу.";

return View();

}

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

HttpContext.Session.SetInt32("Step", 2);

}

else if (step == 2)

{

if (string.IsNullOrEmpty(password))

{

ViewData["Error"] = "Введіть пароль.";

return View();

}

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

string? savedAddress = HttpContext.Session.GetString("Address");

var users = \_context.Users

.Where(u => u.Address == savedAddress && u.Password == password)

ViewData["Users"] = users;

HttpContext.Session.Clear();

}

ViewData["Step"] = HttpContext.Session.GetInt32("Step") ?? 1;

return View();

}

}

}

Клас SingleMethod

Клас **SingleMethod** реалізує контролер, що обробляє логіку для взаємодії з користувачем через один метод. Він працює з сесією для збереження даних, приймає параметри від користувача, перевіряє умови та оновлює стан системи.

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

namespace Task\_3.Controllers

{

public class HomeController : Controller

{

private const string SessionKey = "TriangleSides";

public IActionResult Index(double? side, bool reset = false)

{

var sidesJson = HttpContext.Session.GetString(SessionKey);

var sides = !string.IsNullOrEmpty(sidesJson)

? JsonConvert.DeserializeObject<List<double>>(sidesJson) ?? new List<double>()

: new List<double>();

if (reset)

{

HttpContext.Session.Remove(SessionKey);

return RedirectToAction("Index");

}

if (side.HasValue && side > 0 && sides.Count < 3)

{

sides.Add(side.Value);

HttpContext.Session.SetString(SessionKey, JsonConvert.SerializeObject(sides));

}

ViewBag.Sides = sides;

ViewBag.MaxSide = sides.Count == 3 ? sides.Max() : (double?)null;

return View();

}

}

}

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Task\_6.Data;

using Task\_6.Models;

namespace Task\_6.Controllers

{

public class HomeController : Controller

{

public HomeController(ApplicationDbContext context)

{

\_context = context;

}

public IActionResult Index(int step = 1, string? input = null)

{

if (!string.IsNullOrEmpty(input))

{

if (string.IsNullOrEmpty(input))

{

ViewData["Error"] = "Це поле не може бути порожнім.";

}

else

{

switch (step)

{

case 1:

TempData["Address"] = input;

break;

case 2:

TempData["Password"] = input;

break;

case 3:

TempData["Login"] = input;

break;

case 4:

TempData["Email"] = input;

var user = new User

{

Address = TempData["Address"] as string ?? string.Empty,

Password = TempData["Password"] as string ?? string.Empty,

Login = TempData["Login"] as string ?? string.Empty,

Email = TempData["Email"] as string ?? string.Empty

};

try

{

\_context.Users.Add(user);

\_context.SaveChanges();

return RedirectToAction("Success");

}

catch (DbUpdateException ex)

{

var innerException = ex.InnerException?.Message ?? "No inner exception";

ViewData["Error"] = $"Не вдалося зберегти дані користувача: {innerException}";

}

break;

}

}

return RedirectToAction("Index", new { step = step + 1 });

}

ViewData["Step"] = step;

return View();

}

public IActionResult Success()

{

return View();

}

}

}

using Microsoft.AspNetCore.Mvc;

using Task\_9.Data;

namespace Task\_9.Controllers

{

public class HomeController : Controller

{

public HomeController(AppDbContext context)

{

\_context = context;

}

[HttpGet, HttpPost]

public IActionResult SearchAccount(string? address, string? password)

{

int step = HttpContext.Session.GetInt32("Step") ?? 1;

ViewData["Step"] = step;

if (Request.Method == "POST")

{

if (step == 1)

{

if (string.IsNullOrEmpty(address))

{

ViewData["Error"] = "Введіть адресу.";

}

else

{

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

HttpContext.Session.SetInt32("Step", 2);

ViewData["Step"] = 2;

}

}

else if (step == 2)

{

if (string.IsNullOrEmpty(password))

{

ViewData["Error"] = "Введіть пароль.";

}

else

{

string? savedAddress = HttpContext.Session.GetString("Address");

var users = \_context.Users

.Where(u => u.Address == savedAddress && u.Password == password)

.ToList();

ViewData["Users"] = users;

HttpContext.Session.Clear();

}

}

}

return View("Index");

}

}

}

Висновок

У ході виконання цієї лабораторної роботи я ознайомився з використанням рівнів доступності (scopes) даних для обміну інформацією між контролерами та поданнями у MVC WEB-застосунку. Було розглянуто різні підходи до збереження та передачі даних, зокрема використання TempData, ViewBag і сесійних змінних.

У процесі виконання роботи я реалізував наступні компоненти:

– **MaxTriangleSide** – клас, що отримує дані про сторони трикутника з різних контролерів через TempData, обробляє їх та передає максимальне значення у ViewBag для відображення.

– **AccountRegistration** – клас, що відповідає за покрокову реєстрацію користувача. Було реалізовано перевірку введених даних, їх тимчасове збереження в TempData та остаточне додавання облікового запису до бази даних.

– **AccountSearch** – клас, що здійснює пошук облікового запису користувача за адресою та паролем. Було впроваджено механізм тимчасового збереження даних між запитами та перевірку правильності введеної інформації.

– **SingleClass** – клас, що виконує всі необхідні операції в межах одного контролера, спрощуючи обробку даних.

– **SingleMethod** – контролер, що використовує сесію для збереження даних та обробляє взаємодію з користувачем через один метод.

Ця робота допомогла мені глибше зрозуміти принципи роботи з рівнями доступності даних у MVC-застосунках, а також навчитися ефективно використовувати механізми передачі інформації між контролерами та поданнями. Отримані знання сприятимуть подальшій розробці веб-додатків, що потребують гнучкої взаємодії з даними.