Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Комп’ютерних наук та інформаційних технологій"

**Звіт**

**Лабораторна робота 9**

з дисципліни "Проєктування серверних застосунків"

Виконав: студент групи КН-922В

Мельник Б. В.

Перевірив:

Івашко А. В.

Харків 2025

**Зміст**

[Завдання 1 4](#_Toc197122743)

[Текст програми 4](#_Toc197122744)

[Результати роботи програми 24](#_Toc197122745)

[Алгоритм побудови коду 27](#_Toc197122746)

[Функціональність 28](#_Toc197122747)

[Висновок 29](#_Toc197122748)

[Завдання 2 30](#_Toc197122749)

[Текст програми 30](#_Toc197122750)

[Результати роботи програми 35](#_Toc197122751)

[Алгоритм побудови кодів 36](#_Toc197122752)

[Функціональність 37](#_Toc197122753)

[Висновок 38](#_Toc197122754)

[Завдання 3 39](#_Toc197122755)

[Текст програми 39](#_Toc197122756)

[Результати роботи програми 40](#_Toc197122757)

[Алгоритм побудови кодів 42](#_Toc197122758)

[Функціональність 43](#_Toc197122759)

[Висновок 43](#_Toc197122760)

[Завдання 4 44](#_Toc197122761)

[Текст програми 44](#_Toc197122762)

[Результати роботи програми 46](#_Toc197122763)

[Алгоритм побудови кодів 46](#_Toc197122764)

[Функціональність 47](#_Toc197122765)

[Висновок 47](#_Toc197122766)

[Завдання 5 48](#_Toc197122767)

[Текст програми 48](#_Toc197122768)

[Результати роботи програми 52](#_Toc197122769)

[Алгоритм побудови кодів 54](#_Toc197122770)

[Функціональність 55](#_Toc197122771)

[Висновок 55](#_Toc197122772)

[Завдання 6 56](#_Toc197122773)

[Текст програми 56](#_Toc197122774)

[Результати роботи програми 65](#_Toc197122775)

[Алгоритм побудови кодів 67](#_Toc197122776)

[Функціональність 68](#_Toc197122777)

[Висновок 68](#_Toc197122778)

[Загальний Висновок 69](#_Toc197122779)

**Тема:** Створення завершених MVC WEB-застосунків із обов'язковим стандарт-ним функціоналом, використовуючи можливості ASP.NET. Частина 1.

**Мета роботи:** придбати практику та досвід у створенні WEB-застосунків із стандартним набором сучасних функцій та можливостей за допомогою засо-бів ASP.NET.

**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**

Модифікувати проект останнього (4) завдання минулої роботи, до-дадавши до нього профільну форму користувача умовної системи із використанням дизайну реєстраційної та авторизаційної форм і роз-ташованими на ній реєстраційними (профільними) параметрами ко-ристувача, яка повинна мати можливість змінювати ці параметри. Особливості зміни параметрів визначаються згідно до власного варіанту.

|  |  |
| --- | --- |
| **№ варіанту** | **Особливості зміни реєстраційних параметрів** |
| 12 | Кожен профільний параметр змінюється на власній окремій формі |

## **Текст програми:**

**Views/Account/EditEmail.cshtml:**

@model Task\_1.Models.EditEmailViewModel

@{

ViewData["Title"] = "Edit Email";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Редагувати Email</h2>

<form asp-action="EditEmail" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="Email" class="error-message"></span>

<label asp-for="Email">New Email</label>

<input asp-for="Email" class="form-control" />

</div>

<button type="submit" class="btn-register">Save</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Views/Account/EditName.cshtml:**

@model Task\_1.Models.EditNameViewModel

@{

ViewData["Title"] = "Edit Name";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Редагувати ім’я</h2>

<form asp-action="EditName" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="Name" class="error-message"></span>

<label asp-for="Name">New Name</label>

<input asp-for="Name" class="form-control" />

</div>

<button type="submit" class="btn-register">Save</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Views/Account/EditPassword.cshtml:**

@model Task\_1.Models.EditPasswordViewModel

@{

ViewData["Title"] = "Edit Password";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Редагувати пароль</h2>

<form asp-action="EditPassword" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="NewPassword" class="error-message"></span>

<label asp-for="NewPassword">New Password</label>

<input asp-for="NewPassword" type="password" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="ConfirmPassword" class="error-message"></span>

<label asp-for="ConfirmPassword">Confirm Password</label>

<input asp-for="ConfirmPassword" type="password" class="form-control" />

</div>

<button type="submit" class="btn-register">Save</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Views/Account/EditPhone.cshtml:**

@model Task\_1.Models.EditPhoneViewModel

@{

ViewData["Title"] = "Edit Phone";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Редагувати телефон</h2>

<form asp-action="EditPhone" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="Phone" class="error-message"></span>

<label asp-for="Phone">New Phone</label>

<input asp-for="Phone" class="form-control" />

</div>

<button type="submit" class="btn-register">Save</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Views/Account/Login.cshtml:**

@model Task\_1.Models.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Login</h2>

<form asp-action="Login" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="Name" class="error-message"></span>

<label asp-for="Name">Name</label>

<input asp-for="Name" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="Password" class="error-message"></span>

<label asp-for="Password">Password</label>

<input asp-for="Password" type="password" class="form-control" />

</div>

<span asp-validation-summary="ModelOnly" class="text-danger mb-3 d-block"></span>

<button type="submit" class="btn-register">LOGIN</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Views/Account/Profile.cshtml:**

@model Task\_1.Models.User

@{

ViewData["Title"] = "Profile";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Мій профіль</h2>

<ul class="list-unstyled">

<li><strong>Name:</strong> @Model.Name <a asp-action="EditName" class="ms-2">Edit</a></li>

<li><strong>Email:</strong> @Model.Email <a asp-action="EditEmail" class="ms-2">Edit</a></li>

<li><strong>Phone:</strong> @Model.Phone <a asp-action="EditPhone" class="ms-2">Edit</a></li>

<li><strong>Password:</strong> \*\*\*\*\*\*\*\* <a asp-action="EditPassword" class="ms-2">Edit</a></li>

</ul>

</div>

**Views/Account/Register.cshtml:**

@model Task\_1.Models.RegisterViewModel

@{

ViewData["Title"] = "Register";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Register</h2>

<form asp-action="Register" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="Name" class="error-message"></span>

<label asp-for="Name">Name</label>

<input asp-for="Name" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="Email" class="error-message"></span>

<label asp-for="Email">Email</label>

<input asp-for="Email" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="Phone" class="error-message"></span>

<label asp-for="Phone">Phone</label>

<input asp-for="Phone" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="Password" class="error-message"></span>

<label asp-for="Password">Password</label>

<input asp-for="Password" type="password" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="ConfirmPassword" class="error-message"></span>

<label asp-for="ConfirmPassword">Confirm Password</label>

<input asp-for="ConfirmPassword" type="password" class="form-control" />

</div>

<button type="submit" class="btn-register">REGISTER</button>

</form>

</div>

@section Scripts {

@await Html.PartialAsync("\_ValidationScriptsPartial")

<script>

$(function () {

var form = $('form');

form.removeData('validator').removeData('unobtrusiveValidation');

$.validator.setDefaults({

showErrors: function (errorMap, errorList) {

this.defaultShowErrors();

$('.error-message').each(function () {

var txt = $(this).text().trim();

if (txt) {

var parts = txt.split(' ');

$(this).html(parts.join('<br/>'));

}

});

}

});

$.validator.unobtrusive.parse(form);

});

</script>

}

**Models/EditEmailViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class EditEmailViewModel

{

[Required(ErrorMessage = "Введіть новий Email")]

[EmailAddress(ErrorMessage = "Невірний формат Email")]

public string Email { get; set; } = null!;

}

}

**Models/EditNameViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class EditNameViewModel

{

[Required(ErrorMessage = "Введіть нове ім’я")]

[StringLength(15, ErrorMessage = "Максимум 15 символів")]

[RegularExpression("^[A-Za-z0-9]+$", ErrorMessage = "Лише латиниця та цифри")]

public string Name { get; set; } = null!;

}

}

**Models/EditPasswordViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class EditPasswordViewModel

{

[Required(ErrorMessage = "Введіть новий пароль")]

[MinLength(12, ErrorMessage = "Мінімум 12 символів")]

[RegularExpression(@"^(?=.\*[!@#$%^&\*()]).+$", ErrorMessage = "Потрібен спецсимвол")]

[DataType(DataType.Password)]

public string NewPassword { get; set; } = null!;

[Required(ErrorMessage = "Підтвердіть пароль")]

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

[DataType(DataType.Password)]

public string ConfirmPassword { get; set; } = null!;

}

}

**Models/EditPhoneViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class EditPhoneViewModel

{

[Required(ErrorMessage = "Введіть новий телефон")]

[Phone(ErrorMessage = "Невірний формат телефону")]

public string Phone { get; set; } = null!;

}

}

**Models/LoginViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class LoginViewModel

{

[Required(ErrorMessage = "Введіть ім’я")]

public string Name { get; set; } = null!;

[Required(ErrorMessage = "Введіть пароль")]

[DataType(DataType.Password)]

public string Password { get; set; } = null!;

}

}

**Models/RegisterViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_1.Models

{

public class RegisterViewModel

{

[Required(ErrorMessage = "Введіть ім’я")]

[StringLength(15, ErrorMessage = "Максимум 15 символів")]

[RegularExpression("^[A-Za-z0-9]+$", ErrorMessage = "Лише латиниця та цифри")]

public string Name { get; set; } = null!;

[Required(ErrorMessage = "Введіть Email")]

[EmailAddress(ErrorMessage = "Невірний формат Email")]

public string Email { get; set; } = null!;

[Required(ErrorMessage = "Введіть телефон")]

[Phone(ErrorMessage = "Невірний формат телефону")]

public string Phone { get; set; } = null!;

[Required(ErrorMessage = "Введіть пароль")]

[MinLength(12, ErrorMessage = "Мінімум 12 символів")]

[RegularExpression(@"^(?=.\*[!@#$%^&\*()]).+$", ErrorMessage = "Потрібен спецсимвол")]

[DataType(DataType.Password)]

public string Password { get; set; } = null!;

[Required(ErrorMessage = "Підтвердіть пароль")]

[Compare("Password", ErrorMessage = "Паролі не збігаються")]

[DataType(DataType.Password)]

public string ConfirmPassword { get; set; } = null!;

}

}

**Models/User.cs:**

namespace Task\_1.Models

{

public class User

{

public int Id { get; set; } // Унікальний ідентифікатор

public string Name { get; set; } = null!; // Ім’я користувача

public string Email { get; set; } = null!; // Email користувача

public string Phone { get; set; } = null!; // Телефон користувача

public string Password { get; set; } = null!; // Пароль користувача

}

}

**Data/ ApplicationDbContext.cs:**

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using Task\_1.Models;

namespace Task\_1.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> opts)

: base(opts)

{

}

public DbSet<User> Users { get; set; } = null!; // Колекція користувачів

}

}

**Controllers/AccountController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using Task\_1.Data;

using Task\_1.Models;

using System.Linq;

namespace Task\_1.Controllers

{

public class AccountController : Controller

{

private readonly ApplicationDbContext \_db;

public AccountController(ApplicationDbContext db) => \_db = db;

// REGISTER

[HttpGet]

public IActionResult Register() => View();

[HttpPost]

public IActionResult Register(RegisterViewModel vm)

{

if (\_db.Users.Any(u => u.Name == vm.Name))

ModelState.AddModelError(nameof(vm.Name), "Це ім’я вже зайняте");

if (!ModelState.IsValid)

return View(vm);

\_db.Users.Add(new User

{

Name = vm.Name,

Email = vm.Email,

Phone = vm.Phone,

Password = vm.Password

});

\_db.SaveChanges();

return RedirectToAction("Login");

}

// LOGIN

[HttpGet]

public IActionResult Login() => View();

[HttpPost]

public IActionResult Login(LoginViewModel vm)

{

if (!ModelState.IsValid)

return View(vm);

var user = \_db.Users

.FirstOrDefault(u => u.Name == vm.Name && u.Password == vm.Password);

if (user == null)

{

ModelState.AddModelError(string.Empty, "Невірні облікові дані");

return View(vm);

}

HttpContext.Session.SetString("UserName", user.Name);

return RedirectToAction("Profile");

}

// LOGOUT

public IActionResult Logout()

{

HttpContext.Session.Clear();

return RedirectToAction("Login");

}

// PROFILE

[HttpGet]

public IActionResult Profile()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

var user = \_db.Users.FirstOrDefault(u => u.Name == username);

if (user == null)

return RedirectToAction("Login");

return View(user);

}

// EDIT NAME

[HttpGet]

public IActionResult EditName()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

var user = \_db.Users.First(u => u.Name == username);

var vm = new EditNameViewModel { Name = user.Name };

return View(vm);

}

[HttpPost]

public IActionResult EditName(EditNameViewModel vm)

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

if (\_db.Users.Any(u => u.Name == vm.Name && u.Name != username))

ModelState.AddModelError(nameof(vm.Name), "Це ім’я вже зайняте");

if (!ModelState.IsValid)

return View(vm);

var user = \_db.Users.First(u => u.Name == username);

user.Name = vm.Name;

\_db.SaveChanges();

// Оновлюємо сессію

HttpContext.Session.SetString("UserName", vm.Name);

return RedirectToAction("Profile");

}

// EDIT EMAIL

[HttpGet]

public IActionResult EditEmail()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

var user = \_db.Users.First(u => u.Name == username);

var vm = new EditEmailViewModel { Email = user.Email };

return View(vm);

}

[HttpPost]

public IActionResult EditEmail(EditEmailViewModel vm)

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

if (!ModelState.IsValid)

return View(vm);

var user = \_db.Users.First(u => u.Name == username);

user.Email = vm.Email;

\_db.SaveChanges();

return RedirectToAction("Profile");

}

// EDIT PHONE

[HttpGet]

public IActionResult EditPhone()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

var user = \_db.Users.First(u => u.Name == username);

var vm = new EditPhoneViewModel { Phone = user.Phone };

return View(vm);

}

[HttpPost]

public IActionResult EditPhone(EditPhoneViewModel vm)

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

if (!ModelState.IsValid)

return View(vm);

var user = \_db.Users.First(u => u.Name == username);

user.Phone = vm.Phone;

\_db.SaveChanges();

return RedirectToAction("Profile");

}

// EDIT PASSWORD

[HttpGet]

public IActionResult EditPassword()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

return View(new EditPasswordViewModel());

}

[HttpPost]

public IActionResult EditPassword(EditPasswordViewModel vm)

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

return RedirectToAction("Login");

if (!ModelState.IsValid)

return View(vm);

var user = \_db.Users.First(u => u.Name == username);

user.Password = vm.NewPassword;

\_db.SaveChanges();

return RedirectToAction("Profile");

}

}

}

**\_Layout.cshtml:**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>@ViewData["Title"] – Task\_1</title>

<!-- Bootstrap 5 -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" />

<!-- Animate.css -->

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css" />

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

</head>

<body>

@{

var userName = Context.Session.GetString("UserName");

}

<nav class="navbar navbar-expand-lg navbar-light bg-light mb-4">

<div class="container-fluid">

<a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">Lab\_12</a>

<div class="collapse navbar-collapse">

<ul class="navbar-nav me-auto mb-2 mb-lg-0">

@if (string.IsNullOrEmpty(userName))

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Register">Register</a>

</li>

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Login">Login</a>

</li>

}

else

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Profile">Profile</a>

</li>

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Logout">Logout</a>

</li>

}

</ul>

<span class="navbar-text">

@userName

</span>

</div>

</div>

</nav>

<div class="d-flex justify-content-center align-items-center min-vh-100">

@RenderBody()

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js"></script>

@RenderSection("Scripts", required: false)

</body>

</html>

**site.css:**

/\* фон сторінки \*/

body {

background: url('https://images7.alphacoders.com/116/1162515.jpg') no-repeat center center fixed;

background-size: cover;

}

/\* стилі для картки форми \*/

.form-card {

background: #fff;

padding: 2rem;

border-radius: 8px;

width: 100%;

max-width: 480px;

box-shadow: 0 4px 16px rgba(0,0,0,0.1);

}

.form-card h2 {

margin-bottom: 1.5rem;

font-weight: 500;

}

/\* стилі для полів вводу \*/

.form-material .form-control {

border: none;

border-bottom: 1px solid #ccc;

border-radius: 0;

background: transparent;

box-shadow: none;

}

.form-material .form-control:focus {

outline: none;

border-bottom-color: #1abc9c;

}

/\* стилі для лейблів \*/

.form-material label {

display: block;

color: #888;

font-size: 0.9rem;

margin-top: 0.25rem;

margin-bottom: 0.25rem;

}

/\* стилі для повідомлень про помилки \*/

.error-message {

display: block;

margin-bottom: 0.25rem;

color: #e74c3c;

font-size: 0.8rem;

line-height: 1.2;

animation: fade-in 0.4s;

}

/\* стилі для кнопки \*/

.btn-register {

width: 30%;

background-color: #1abc9c;

border: none;

padding: .75rem;

text-transform: uppercase;

letter-spacing: 1px;

}

.btn-register:hover {

background-color: #16a085;

}

/\* анімація для плавної появи \*/

@keyframes fade-in {

from {

opacity: 0;

}

to {

opacity: 1;

}

}

## **Результати роботи програми:**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.1. Вигляд БД

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.  
Рисунок 1.2. Вигляд форми Profile

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.3. Редагування Ім’я на окремій формі

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.4. Редагування Email на окремій формі

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.5. Редагування Phone на окремій формі

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.6. Редагування Паролю на окремій формі

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.7. Вигляд відредагованого Profile

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.8. Оновлений вміст БД

## **Алгоритм побудови коду**

1. **Створення або відкриття базового проєкту**  
   – Відкрити останню версію проєкту з минулого завдання (Task\_4).  
   – Переконатися, що в Startup/Program налаштована сесія та EF Core.
2. **Додавання моделі користувача та контексту**  
   – У папці Models створити клас User з полями Id, Name, Email, Phone, Password.  
   – У папці Data відредагувати ApplicationDbContext, додавши DbSet<User> Users.
3. **Реєстрація та аутентифікація**  
   – Додати у AccountController дії Register (GET/POST) і Login (GET/POST).  
   – У методах POST перевіряти унікальність і коректність даних через ModelState.  
   – Після успішного входу зберігати ім’я в сесії (HttpContext.Session).
4. **Побудова профільної частини**  
   – Додати дію Profile (GET) в AccountController, яка читає поточного користувача з бази за іменем із сесії.  
   – У макеті \_Layout.cshtml додати пункти меню Profile/Logout для залогіненого користувача.
5. **Розбиття зміни параметрів на окремі форми**  
   – Для кожного параметра (Name, Email, Phone, Password) створити ViewModel-класи:  
   EditNameViewModel, EditEmailViewModel, EditPhoneViewModel, EditPasswordViewModel з атрибутами валідації.  
   – У AccountController реалізувати дії EditName, EditEmail, EditPhone, EditPassword (GET/POST).  
   – У POST-методах перевіряти ModelState, оновлювати відповідне поле в базі та зберігати зміни.
6. **Створення та стилізація виглядів (Views)**  
   – Для кожної дії додати Razor-сторінки:
   * Views/Account/Register.cshtml, Login.cshtml, Profile.cshtml
   * EditName.cshtml, EditEmail.cshtml, EditPhone.cshtml, EditPassword.cshtml  
     – Підключити спільні скрипти валідації (\_ValidationScriptsPartial) і налаштувати адаптивні стилі в site.css.
7. **Клієнт- та серверна валідація**  
   – Налаштувати jQuery Validation Unobtrusive, перевизначити showErrors для виводу помилок по рядках.  
   – Переконатися, що кожне поле перевіряється і помилка відображається саме під відповідним полем.
8. **Тестування**  
   – Провести кроки: реєстрація → вхід → перегляд профілю → зміна кожного параметра → вихід.  
   – Перевірити обробку некоректних даних та повернення на ту ж форму з повідомленнями про помилки.

## **Функціональність**

* **Реєстрація користувача** з перевіркою унікальності і коректності введених даних (ім’я, email, телефон, пароль з підтвердженням).
* **Вхід (Login)** із перевіркою імені та пароля; при успіху—запис імені в сесію.
* **Профіль користувача**: відображення поточних значень полів і кнопки «Edit» біля кожного параметра.
* **Окремі форми редагування**: кожен параметр змінюється на власній сторінці з клієнтською та серверною валідацією.
* **Навігація**: у шапці сайту динамічно відображаються пункти меню залежно від стану сесії (Register/Login або Profile/Logout).

## **Висновок**

Виконана модифікація вихідного проєкту забезпечила зручний інтерфейс профілю з окремими формами для зміни кожного параметра згідно варіанта №12. Завдяки поєднанню клієнтської та серверної валідації реалізовано надійну перевірку даних, а чітка навігація та сучасний дизайн сприяють покращенню користувацького досвіду. Ця структура легко масштабується під додавання нових полів чи ролей у майбутньому.

# **Завдання 2**

Додати до проекту форму відправки повідомлень користувачів умо-вної системи із використанням дизайну форм минулих завдань, яка повинна мати поля ТО для визначення користувача-отримувача по-відомлення, ТHEME та ТЕХТ для визначення теми і тексту повідомлення відповідно. Повідомлення мають зберігатись у відповідному файлі повідомлень для форми отримання цих повідомлень.

## **Текст програми:**

**View/Messages/Send.cshtml:**

@model Task\_2.Models.SendMessageViewModel

@{

ViewData["Title"] = "Send Message";

var users = ViewBag.Users as List<string>;

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Відправити повідомлення</h2>

<form asp-action="Send" method="post" novalidate>

<div class="form-material mb-4">

<span asp-validation-for="To" class="error-message"></span>

<label asp-for="To">To</label>

<select asp-for="To" class="form-control">

<option value="">-- Оберіть отримувача --</option>

@foreach (var u in users)

{

<option value="@u">@u</option>

}

</select>

</div>

<div class="form-material mb-4">

<span asp-validation-for="Theme" class="error-message"></span>

<label asp-for="Theme">Theme</label>

<input asp-for="Theme" class="form-control" />

</div>

<div class="form-material mb-4">

<span asp-validation-for="Text" class="error-message"></span>

<label asp-for="Text">Text</label>

<textarea asp-for="Text" class="form-control" rows="5"></textarea>

</div>

<button type="submit" class="btn-register">Send</button>

</form>

</div>

**View/Messages/Sent.cshtml:**

@model List<Task\_2.Models.Message>

@{

ViewData["Title"] = "Sent Messages";

}

<div class="form-card animate\_\_animated animate\_\_fadeIn">

<h2>Надіслані повідомлення</h2>

@if (!Model.Any())

{

<p>Ви ще не надсилали жодного повідомлення.</p>

}

else

{

<ul class="list-group">

@foreach (var m in Model)

{

<li class="list-group-item">

<strong>@m.SentAt.ToString("g")</strong> до <strong>@m.To</strong><br />

<em>@m.Theme</em><br />

<p>@m.Text</p>

</li>

}

</ul>

}

</div>

**Models/SendMessageViewModel.cs:**

using System.ComponentModel.DataAnnotations;

namespace Task\_2.Models

{

public class SendMessageViewModel

{

[Required(ErrorMessage = "Оберіть отримувача")]

public string To { get; set; } = null!;

[Required(ErrorMessage = "Введіть тему")]

public string Theme { get; set; } = null!;

[Required(ErrorMessage = "Введіть текст повідомлення")]

public string Text { get; set; } = null!;

}

}

**Controllers/MessagesController.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using Task\_2.Data;

using Task\_2.Models;

using System.Text.Json;

namespace Task\_2.Controllers

{

public class MessagesController : Controller

{

private readonly ApplicationDbContext \_db;

private readonly string \_filePath;

public MessagesController(ApplicationDbContext db, IWebHostEnvironment env)

{

\_db = db;

\_filePath = Path.Combine(env.ContentRootPath, "App\_Data", "messages.json");

var dir = Path.GetDirectoryName(\_filePath)!;

if (!Directory.Exists(dir))

Directory.CreateDirectory(dir);

if (!System.IO.File.Exists(\_filePath))

System.IO.File.WriteAllText(\_filePath, "[]");

}

[HttpGet]

public IActionResult Send()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

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

ViewBag.Users = \_db.Users.Select(u => u.Name).ToList();

return View(new SendMessageViewModel());

}

[HttpPost]

public IActionResult Send(SendMessageViewModel vm)

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

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

ViewBag.Users = \_db.Users.Select(u => u.Name).ToList();

if (!ModelState.IsValid)

return View(vm);

var json = System.IO.File.ReadAllText(\_filePath);

var messages = JsonSerializer.Deserialize<List<Message>>(json) ?? new List<Message>();

messages.Add(new Message

{

From = username,

To = vm.To,

Theme = vm.Theme,

Text = vm.Text,

SentAt = DateTime.Now

});

System.IO.File.WriteAllText(\_filePath,

JsonSerializer.Serialize(messages, new JsonSerializerOptions { WriteIndented = true }));

return RedirectToAction("Sent");

}

[HttpGet]

public IActionResult Sent()

{

var username = HttpContext.Session.GetString("UserName");

if (string.IsNullOrEmpty(username))

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

var json = System.IO.File.ReadAllText(\_filePath);

var messages = JsonSerializer.Deserialize<List<Message>>(json) ?? new List<Message>();

var sent = messages

.Where(m => m.From == username)

.OrderByDescending(m => m.SentAt)

.ToList();

return View(sent);

}

}

}

**App\_Data/messages.json:**

[

{

"From": "Bohdan",

"To": "Bogdan10",

"Theme": "How are you?",

"Text": "Hello! Let\u0027s go to the sea tomorrow!",

"SentAt": "2025-04-28T15:40:59.7980273+03:00"

},

{

"From": "Bogdan100",

"To": "NeMo",

"Theme": "How are you?",

"Text": "Hello! Let\u0027s go to the sea?",

"SentAt": "2025-05-02T21:59:31.1377905+03:00"

}

]

## **Результати роботи програми:**

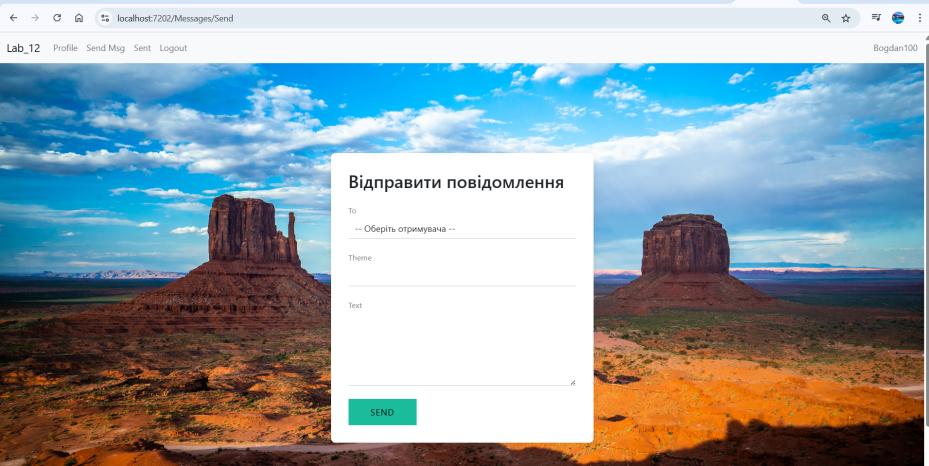


Рисунок 2.1. Вигляд форми відправлення повідомлень

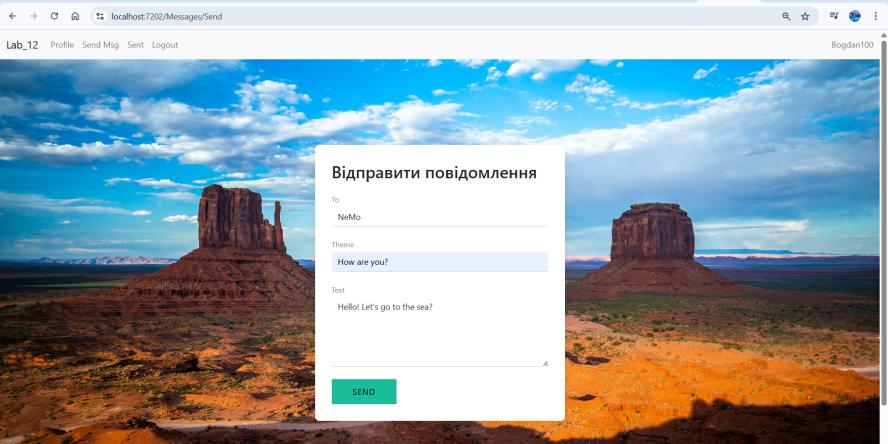


Рисунок 2.2. Заповнення форми відправки повідомлень

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.3. Вигляд надісланого повідомлення

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.4. Виконання збереження надісланих повідомлень у файлі messages.json

## **Алгоритм побудови кодів**

1. **Створення моделі повідомлення**  
   – Додати клас Message у папку Models з полями From, To, Theme, Text, SentAt.
2. **Підготовка сховища файлів**  
   – У конструкторі MessagesController визначити шлях до App\_Data/messages.json.  
   – Переконатися, що папка та файл існують (створити їх за потреби, ініціалізувати порожнім масивом []).
3. **ViewModel для відправки**  
   – Створити SendMessageViewModel з властивостями To, Theme, Text й атрибутами [Required].
4. **Контролер повідомлень**  
   – У MessagesController написати:
   * Send (GET) — підвантажити список користувачів у ViewBag.Users і показати порожню форму.
   * Send (POST) — перевірити ModelState, десеріалізувати повідомлення з файлу, додати нове (з полем From із сесії та SentAt = DateTime.Now), серіалізувати назад і перенаправити на Sent.
   * Sent (GET) — прочитати всі повідомлення, відфільтрувати лише ті, що відправлені поточним користувачем, відсортувати за датою й передати у View.
5. **Створення виглядів (Views)**  
   – Views/Messages/Send.cshtml — форма з полем select для To, полями Theme і Text, скрипти валідації та адаптивний дизайн.  
   – Views/Messages/Sent.cshtml — список відправлених повідомлень (відображення дати, отримувача, теми та тексту).
6. **Клієнтська та серверна валідація**  
   – Підключити \_ValidationScriptsPartial, додати стилі .error-message.  
   – Переконатися, що порожні або некоректні поля блокуються.
7. **Тестування**  
   – Зареєструватися, зайти, відправити кілька повідомлень на різних користувачів, перевірити їхнього отримувача в Sent.  
   – Перевірити, що файл messages.json справді містить коректний JSON із записами.

## **Функціональність**

* **Відправка повідомлення**: користувач обирає отримувача, вводить тему й текст, натискає «Send».
* **Збереження**: кожне повідомлення додається до App\_Data/messages.json у форматі масиву JSON.
* **Перегляд надісланих**: окрема сторінка показує всі повідомлення, відправлені поточним користувачем, у зворотному хронологічному порядку.

## **Висновок**

Реалізація Task\_2 дозволила додати до системи зручний інтерфейс для обміну повідомленнями між користувачами із збереженням історії у файлі. Використання JSON-сховища забезпечує простоту розгортання й легку масштабованість, а вбудована валідація гарантує коректність введених даних.

# **Завдання 3**

Додати до проекту форму отримання повідомлень користувачів умо-вної системи із використанням дизайну форм минулих завдань, яка повинна мати перелік повідомлень для поточного користувача із можливістю перегляду кожного повідомлення в окремій формі.

## **Текст програми:**

**Inbox.cshtml:**

@model List<Task\_3.Models.Message>

@{

ViewData["Title"] = "Inbox";

}

<div class="form-card animate\_\_fadeIn">

<h2>Вхідні повідомлення</h2>

@if (!Model.Any())

{

<p>немає</p>

}

else

{

<ul class="list-group">

@foreach (var m in Model)

{

<li class="list-group-item">

<a asp-action="Details" asp-route-id="@m.Id">

<strong>@m.SentAt:g</strong> від <strong>@m.From</strong> — @m.Theme

</a>

</li>

}

</ul>

}

</div>

**Details.cshtml:**

@model Task\_3.Models.Message

@{

ViewData["Title"] = "Message Details";

}

<div class="form-card animate\_\_fadeIn">

<h2>Повідомлення</h2>

<dl class="row">

<dt class="col-sm-3">From</dt>

<dd class="col-sm-9">@Model.From</dd>

<dt class="col-sm-3">To</dt>

<dd class="col-sm-9">@Model.To</dd>

<dt class="col-sm-3">Theme</dt>

<dd class="col-sm-9">@Model.Theme</dd>

<dt class="col-sm-3">Sent At</dt>

<dd class="col-sm-9">@Model.SentAt.ToString("f")</dd>

<dt class="col-sm-3">Text</dt>

<dd class="col-sm-9">@Model.Text</dd>

</dl>

<a asp-action="Inbox" class="btn btn-secondary">Back to Inbox</a>

</div>

## **Результати роботи програми:**

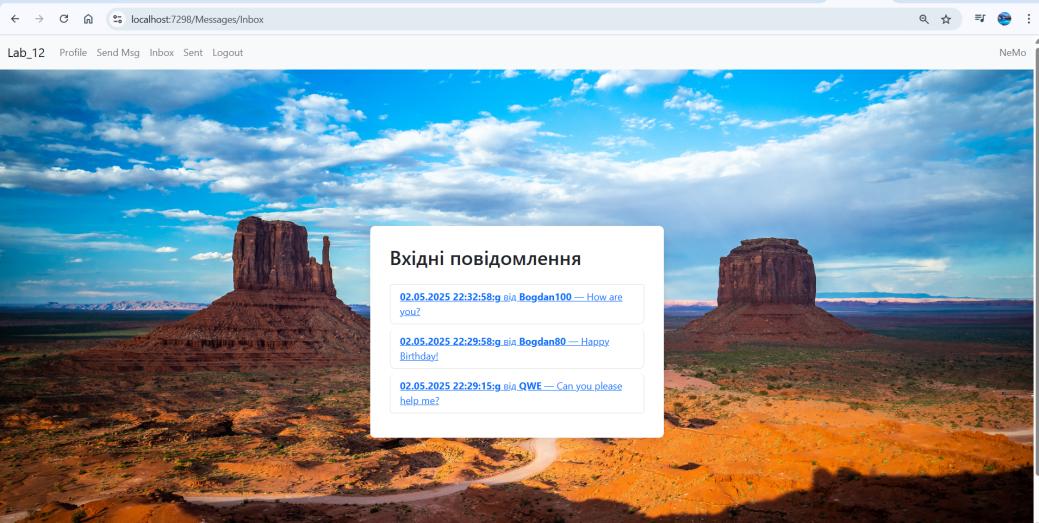


Рисунок 3.1. Вигляд форми вхідних повідомлень

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2. Перегляд вмісту повідомлень на окремій формі

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.3. Перегляд вмісту повідомлень на окремій формі

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.4. Перегляд вмісту повідомлень на окремій формі

## **Алгоритм побудови кодів**

1. **Перевірити модель**
   * У Models/Message.cs має бути поле Id (унікальний ідентифікатор), а також From, To, Theme, Text, SentAt.
2. **Додати дії в контролер**
   * У MessagesController реалізувати дві нові дії:
     + Inbox() (GET) для списку вхідних повідомлень;
     + Details(int id) (GET) для перегляду одного повідомлення.
3. **Реалізація Inbox()**
   * Зчитати з сесії поточне ім’я користувача (username).
   * Прочитати і десеріалізувати файл messages.json у List<Message>.
   * Відфільтрувати повідомлення з m.To == username.
   * Відсортувати за SentAt (спочатку найновіші) та повернути View(filteredList).
4. **Реалізація Details(int id)**
   * Зчитати username із сесії.
   * Прочитати й десеріалізувати весь список повідомлень.
   * Знайти повідомлення з потрібним Id.
   * Перевірити, що m.To == username; якщо ні — перенаправити на Inbox.
   * Повернути View(message).
5. **Створити вигляди**
   * Views/Messages/Inbox.cshtml — список з посиланнями на деталі (asp-route-id="@m.Id").
   * Views/Messages/Details.cshtml — відображення полів повідомлення у вигляді <dl>.
6. **Навігація**
   * Додати в меню або профіль посилання «Inbox», що веде на Messages/Inbox.
7. **Тестування**
   * Перевірити: після логіну користувач бачить лише свої вхідні повідомлення та може відкрити деталізацію кожного.

## **Функціональність**

* **Список вхідних**: показує всі повідомлення, адресовані поточному користувачеві, з датою, відправником і темою.
* **Деталі повідомлення**: окремий екран з полями From, To, Theme, SentAt, Text.
* **Безпека**: доступ до деталей лише для тих повідомлень, що належать поточному користувачу.

## **Висновок**

Додано зручний інтерфейс “Inbox” для виведення та перегляду вхідних повідомлень у системі. Розподіл на список і окрему детальну форму забезпечує чітку навігацію. Використання JSON-файлу як простого сховища дозволяє легко масштабувати функціонал без складного налаштування бази даних.

## **Завдання 4**

Додати до проекту форму заставку із логотипом і назвою умовної системи, з'являється на початку функціонування цієї системи.

## **Текст програми:**

**Views/Splash/ Index.cshtml:**

@{

Layout = null;

}

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>NovaChat</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" />

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css" />

<style>

body {

background: url('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS7\_zlBPsa\_kgNmEOgmIlH3ExsVIVdLrV99DQ&s')

no-repeat center center fixed;

background-size: cover;

}

.splash-container {

position: absolute; top:50%; left:50%;

transform: translate(-50%, -50%);

text-align: center;

}

.splash-logo {

width: 150px; margin-bottom: 20px;

}

.splash-title {

color: #fff;

font-size: 3rem;

font-weight: 700;

text-shadow: 2px 2px 4px rgba(0,0,0,0.5);

}

</style>

</head>

<body>

<div class="splash-container animate\_\_animated animate\_\_fadeIn">

<img src="https://upload.wikimedia.org/wikipedia/commons/thumb/b/be/Facebook\_Messenger\_logo\_2020.svg/2048px-Facebook\_Messenger\_logo\_2020.svg.png"

alt="NovaChat Logo" class="splash-logo animate\_\_animated animate\_\_zoomIn" />

<div class="splash-title animate\_\_animated animate\_\_fadeInUp">NovaChat</div>

</div>

<script>

setTimeout(function() {

window.location.href = '@Url.Action("Login", "Account")';

}, 3000);

</script>

</body>

</html>

**Controllers/SplashController.cs:**

using Microsoft.AspNetCore.Mvc;

namespace Task\_4.Controllers

{

public class SplashController : Controller

{

[HttpGet]

public IActionResult Index() => View();

}

}

## **Результати роботи програми:**

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4.1. Вигляд форми заставки

## **Алгоритм побудови кодів**

1. **Створити View**
   * Додати папку Views/Splash і файл Index.cshtml.
   * Встановити в початку файлу Layout = null;, щоб прибрати загальний шаблон.
2. **Розмітка та стилі**
   * Підключити у <head> Bootstrap і Animate.css.
   * Написати HTML-структуру: контейнер із класом .splash-container, всередині — зображення з класом .splash-logo та заголовок .splash-title.
   * Вбудувати CSS (inline чи у окремому файлі) для фону сторінки, центрованого розташування контейнера та стилізації логотипа і назви.
3. **JS-перенаправлення**
   * Додати внизу <script> з setTimeout, яке через 3000 мс робить window.location.href = '@Url.Action("Login","Account")'.
4. **Контролер**
   * Створити SplashController з дією

[HttpGet]

public IActionResult Index() => View();

1. **Налаштування маршруту**
   * У Program.cs/Startup.cs (залежно від версії) вказати маршрут за замовчуванням на controller = "Splash", action = "Index".
2. **Тестування**
   * Запустити додаток, упевнитися, що першим відкривається Splash-екран і через 3 савтоматично відбувається перехід на сторінку входу.

## **Функціональність**

* **Заставка з логотипом і назвою “NovaChat”**, що з’являється одразу після запуску.
* **Центроване розташування** елементів на фоні зображення із плавними анімаціями (fadeIn, zoomIn, fadeInUp).
* **Автоматичний перехід** на сторінку входу (Account/Login) через 3 секунди.
* **Відсутність навігаційних елементів** і зайвих компонентів за рахунок Layout = null.

## **Висновок**

Реалізація заставки надала додатку професійного вигляду, підкреслила бренд і створила плавний перехід до функціоналу входу, покращивши перше враження користувача та загальний UX.

# **Завдання 5**

Об'єднати зроблені форми за сценарієм функціонування умовної системи, який характеризується наступною схемою.

Изображение выглядит как текст, снимок экрана, Цвет электрик, визитная карточка

Контент, сгенерированный ИИ, может содержать ошибки.

## **Текст програми:**

**Program.cs:**

using Microsoft.EntityFrameworkCore;

using Task\_5.Data;

var builder = WebApplication.CreateBuilder(args);

// DbContext

builder.Services.AddDbContext<ApplicationDbContext>(opts =>

opts.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// MVC + Session

builder.Services.AddControllersWithViews();

builder.Services.AddSession();

builder.Services.AddHttpContextAccessor();

var app = builder.Build();

app.UseStaticFiles();

app.UseSession();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Splash}/{action=Index}/{id?}"

);

app.Run();

**\_Layout.cshtml:**

@{

var ctrl = ViewContext.RouteData.Values["controller"]?.ToString();

var act = ViewContext.RouteData.Values["action"]?.ToString();

var user = Context.Session.GetString("UserName");

}

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>@ViewData["Title"] – NovaChat</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" />

<link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css" />

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

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-light bg-light mb-4">

<div class="container-fluid">

<a class="navbar-brand" asp-controller="Splash" asp-action="Index">NovaChat</a>

<button class="navbar-toggler" type="button"

data-bs-toggle="collapse" data-bs-target="#nav">

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

</button>

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

<ul class="navbar-nav me-auto mb-2 mb-lg-0">

@\* Схема навігації 1→1 \*@

@if (ctrl == "Splash")

{

@\* жодних лінків \*@

}

else if (ctrl == "Account" && act == "Login")

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Register">Register</a>

</li>

}

else if (ctrl == "Account" && act == "Register")

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Login">Login</a>

</li>

}

else if (ctrl == "Account" && act == "Profile")

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Logout">Logout</a>

</li>

<li class="nav-item">

<a class="nav-link" asp-controller="Messages" asp-action="Send">Send Msg</a>

</li>

<li class="nav-item">

<a class="nav-link" asp-controller="Messages" asp-action="Inbox">Inbox</a>

</li>

<li class="nav-item">

<a class="nav-link" asp-controller="Messages" asp-action="Sent">Sent</a>

</li>

}

else if (ctrl == "Messages" && act == "Send")

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Profile">Back to Profile</a>

</li>

}

else if (ctrl == "Messages" && (act == "Inbox" || act == "Sent" || act == "Details"))

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Profile">Back to Profile</a>

</li>

}

else if (ctrl == "Account" && (act.StartsWith("Edit")))

{

<li class="nav-item">

<a class="nav-link" asp-controller="Account" asp-action="Profile">Back to Profile</a>

</li>

}

</ul>

<span class="navbar-text text-muted">@user</span>

</div>

</div>

</nav>

<div class="d-flex justify-content-center align-items-center min-vh-100">

@RenderBody()

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>

<script src="~/lib/jquery/dist/jquery.min.js"></script>

<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js"></script>

@RenderSection("Scripts", required: false)

</body>

</html>

## **Результати роботи програми:**

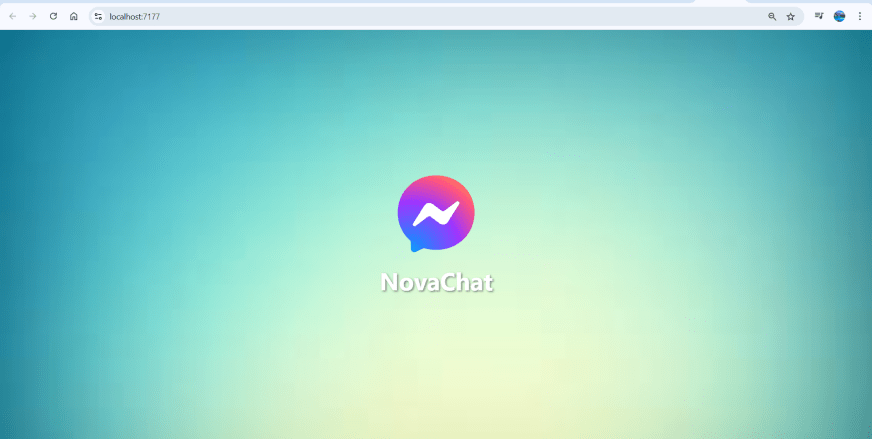


Рисунок 5.1. Форма заставка

Изображение выглядит как облако, небо, снимок экрана, пейзаж

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5.2. Перехід на форму Login

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5.3. Перехід на форму Register

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5.4. Перехід на форму Profile

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5.5. Перехід на форму відправки повідомлень

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5.6. Перехід на форму відправлених повідомлень

Изображение выглядит как облако, небо, снимок экрана, текст

Контент, сгенерированный ИИ, может содержать ошибки.

Перехід на форму отриманих повідомлень

## **Алгоритм побудови кодів**

1. **Налаштування Program.cs**
   * Додати AddDbContext<ApplicationDbContext>() з підключенням SQL Server.
   * Підключити MVC (AddControllersWithViews()), сесію (AddSession()) та доступ до HttpContext (AddHttpContextAccessor()).
   * У app викликати UseStaticFiles(), UseSession() і задати маршрут за замовчуванням на Splash/Index.
2. **Реалізація контролерів і Views**
   * **SplashController** → Index() (View без макету, із заставкою).
   * **AccountController** → дії Register, Login, Profile, Edit… згідно з попередніми завданнями.
   * **MessagesController** → Send (відправка), Inbox (список вхідних), Sent (список надісланих), Details (деталі).
3. **Спільний макет \_Layout.cshtml**
   * Використати @RenderBody() для вставки поточної сторінки.
   * Через змінні ctrl і act задати навігацію 1→1 (заставка → Login ↔ Register → Profile ↔ Send/Inbox/Sent/Details).
   * Виводити ім’я користувача з сесії в шапці.
4. **Оформлення інтерфейсу**
   * Підключити Bootstrap і Animate.css.
   * Ввести в site.css стилі для .form-card, полів, повідомлень про помилки та кнопок.
5. **Тестування кінцевого сценарію**
   * Запуск: Splash → автоматично Login → за потреби Register → Profile → Send → Inbox → Details → Back to Profile → Logout.
   * Перевірити, що кожен перехід відповідає схемі й що захист маршрутів працює (без сесії — перенаправлення на Login).

## **Функціональність**

* **Заставка** з логотипом NovaChat і автоматичним переходом на Login.
* **Реєстрація/Авторизація** користувача із збереженням у сесії.
* **Профіль** із можливістю редагувати кожен параметр на окремих формах.
* **Відправка повідомлень**: вибір отримувача, тема, текст; збереження в messages.json.
* **Перегляд повідомлень**: вхідні (Inbox), надіслані (Sent) та деталі кожного повідомлення.
* **Навігація**: чітке «одна сторінка → одна» згідно зі схемою, без зайвих лінків.

## **Висновок**

У результаті всі форми об’єднано в єдину систему з логічним та інтуїтивним сценарієм користувача. Завдяки централізованому макету та продуманій маршрутизації реалізовано плавний перехід між заставкою, авторизацією, профілем і обміном повідомленнями. Це забезпечує високий рівень UX і полегшує подальший розвиток проєкту.

# **Завдання 6**

Уся інформація, що потрібна для функціонування системи має збері-гатися в текстових або в двійкових файлах за бажанням розробника.

## **Текст програми:**

**App\_Data/messages.json:**

[

{

"Id": "f2edf2bd-59df-4066-926b-ead311d73533",

"From": "Bogdan",

"To": "Test1",

"Theme": "How are you?",

"Text": "Hello!",

"SentAt": "2025-04-29T13:54:05.6950301+03:00"

}

]

**App\_Data/users.json:**

[

{

"Name": "Bogdan",

"Email": "nemofresh8@gmail.com",

"Phone": "\u002B380963945855",

"Password": "12458963Eb@10"

},

{

"Name": "Test1",

"Email": "nemofresh10000@gmail.com",

"Phone": "\u002B380502414589",

"Password": "12458963Eb@78"

}

]

**Services/FileStore.cs:**

using System.Text.Json;

namespace Task\_6.Services

{

public class FileStore<T>

{

private readonly string \_path;

private readonly JsonSerializerOptions \_opts = new() { WriteIndented = true };

public FileStore(IWebHostEnvironment env, string fileName)

{

var dir = Path.Combine(env.ContentRootPath, "App\_Data");

if (!Directory.Exists(dir)) Directory.CreateDirectory(dir);

\_path = Path.Combine(dir, fileName);

if (!File.Exists(\_path)) File.WriteAllText(\_path, "[]");

}

public List<T> Load()

=> JsonSerializer.Deserialize<List<T>>(File.ReadAllText(\_path)) ?? new List<T>();

public void Save(List<T> items)

=> File.WriteAllText(\_path, JsonSerializer.Serialize(items, \_opts));

}

}

**Program.cs:**

using Task\_6.Services;

using Task\_6.Models;

var builder = WebApplication.CreateBuilder(args);

// File-based stores

builder.Services.AddSingleton<FileStore<User>>(sp =>

new FileStore<User>(sp.GetRequiredService<IWebHostEnvironment>(), "users.json"));

builder.Services.AddSingleton<FileStore<Message>>(sp =>

new FileStore<Message>(sp.GetRequiredService<IWebHostEnvironment>(), "messages.json"));

builder.Services.AddControllersWithViews();

builder.Services.AddSession();

builder.Services.AddHttpContextAccessor();

var app = builder.Build();

app.UseStaticFiles();

app.UseSession();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Splash}/{action=Index}/{id?}"

);

app.Run();

**Controllers/AccountController.cs:**

using Microsoft.AspNetCore.Mvc;

using Task\_6.Models;

using Task\_6.Services;

namespace Task\_6.Controllers

{

public class AccountController : Controller

{

private readonly FileStore<User> \_users;

public AccountController(FileStore<User> users) => \_users = users;

[HttpGet]

public IActionResult Register() => View();

[HttpPost]

public IActionResult Register(RegisterViewModel vm)

{

var all = \_users.Load();

if (all.Any(u => u.Name == vm.Name))

ModelState.AddModelError(nameof(vm.Name), "Name taken");

if (!ModelState.IsValid)

return View(vm);

all.Add(new User

{

Name = vm.Name,

Email = vm.Email,

Phone = vm.Phone,

Password = vm.Password

});

\_users.Save(all);

return RedirectToAction("Login");

}

[HttpGet]

public IActionResult Login() => View();

[HttpPost]

public IActionResult Login(LoginViewModel vm)

{

if (!ModelState.IsValid)

return View(vm);

var user = \_users.Load()

.FirstOrDefault(u => u.Name == vm.Name && u.Password == vm.Password);

if (user == null)

{

ModelState.AddModelError(string.Empty, "Invalid credentials");

return View(vm);

}

HttpContext.Session.SetString("UserName", user.Name);

return RedirectToAction("Profile");

}

public IActionResult Logout()

{

HttpContext.Session.Clear();

return RedirectToAction("Login");

}

[HttpGet]

public IActionResult Profile()

{

var name = HttpContext.Session.GetString("UserName");

if (name == null)

return RedirectToAction("Login");

var user = \_users.Load().First(u => u.Name == name);

return View(user);

}

[HttpGet]

public IActionResult EditName()

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

var user = \_users.Load().First(u => u.Name == name);

return View(new EditNameViewModel { Name = user.Name });

}

[HttpPost]

public IActionResult EditName(EditNameViewModel vm)

{

var old = HttpContext.Session.GetString("UserName");

if (old == null) return RedirectToAction("Login");

var all = \_users.Load();

if (all.Any(u => u.Name == vm.Name && u.Name != old))

ModelState.AddModelError(nameof(vm.Name), "Name taken");

if (!ModelState.IsValid)

return View(vm);

var user = all.First(u => u.Name == old);

user.Name = vm.Name;

\_users.Save(all);

HttpContext.Session.SetString("UserName", vm.Name);

return RedirectToAction("Profile");

}

[HttpGet]

public IActionResult EditEmail()

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

var user = \_users.Load().First(u => u.Name == name);

return View(new EditEmailViewModel { Email = user.Email });

}

[HttpPost]

public IActionResult EditEmail(EditEmailViewModel vm)

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

if (!ModelState.IsValid) return View(vm);

var all = \_users.Load();

var user = all.First(u => u.Name == name);

user.Email = vm.Email;

\_users.Save(all);

return RedirectToAction("Profile");

}

[HttpGet]

public IActionResult EditPhone()

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

var user = \_users.Load().First(u => u.Name == name);

return View(new EditPhoneViewModel { Phone = user.Phone });

}

[HttpPost]

public IActionResult EditPhone(EditPhoneViewModel vm)

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

if (!ModelState.IsValid) return View(vm);

var all = \_users.Load();

var user = all.First(u => u.Name == name);

user.Phone = vm.Phone;

\_users.Save(all);

return RedirectToAction("Profile");

}

[HttpGet]

public IActionResult EditPassword()

{

if (HttpContext.Session.GetString("UserName") == null)

return RedirectToAction("Login");

return View();

}

[HttpPost]

public IActionResult EditPassword(EditPasswordViewModel vm)

{

var name = HttpContext.Session.GetString("UserName");

if (name == null) return RedirectToAction("Login");

if (!ModelState.IsValid) return View(vm);

var all = \_users.Load();

var user = all.First(u => u.Name == name);

user.Password = vm.NewPassword;

\_users.Save(all);

return RedirectToAction("Profile");

}

}

}

**Controllers/MessagesController.cs:**

using Microsoft.AspNetCore.Mvc;

using Task\_6.Models;

using Task\_6.Services;

namespace Task\_6.Controllers

{

public class MessagesController : Controller

{

private readonly FileStore<Message> \_msgs;

private readonly FileStore<User> \_users;

public MessagesController(FileStore<Message> msgs, FileStore<User> users)

{

\_msgs = msgs;

\_users = users;

}

[HttpGet]

public IActionResult Send()

{

var from = HttpContext.Session.GetString("UserName");

if (from == null) return RedirectToAction("Login", "Account");

ViewBag.Users = \_users.Load().Select(u => u.Name).ToList();

return View(new SendMessageViewModel());

}

[HttpPost]

public IActionResult Send(SendMessageViewModel vm)

{

var from = HttpContext.Session.GetString("UserName");

if (from == null) return RedirectToAction("Login", "Account");

var list = \_msgs.Load();

list.Add(new Message

{

Id = Guid.NewGuid(),

From = from,

To = vm.To,

Theme = vm.Theme,

Text = vm.Text,

SentAt = DateTime.Now

});

\_msgs.Save(list);

return RedirectToAction("Sent");

}

[HttpGet]

public IActionResult Inbox()

{

var me = HttpContext.Session.GetString("UserName");

if (me == null) return RedirectToAction("Login", "Account");

var inbox = \_msgs.Load()

.Where(m => m.To == me)

.OrderByDescending(m => m.SentAt)

.ToList();

return View(inbox);

}

[HttpGet]

public IActionResult Sent()

{

var me = HttpContext.Session.GetString("UserName");

if (me == null) return RedirectToAction("Login", "Account");

var sent = \_msgs.Load()

.Where(m => m.From == me)

.OrderByDescending(m => m.SentAt)

.ToList();

return View(sent);

}

[HttpGet]

public IActionResult Details(Guid id)

{

var me = HttpContext.Session.GetString("UserName");

if (me == null) return RedirectToAction("Login", "Account");

var msg = \_msgs.Load().FirstOrDefault(m => m.Id == id);

if (msg == null || (msg.From != me && msg.To != me))

return NotFound();

return View(msg);

}

}

}

## **Результати роботи програми:**

Изображение выглядит как облако, текст, небо, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.1. Вигляд даних на формі Profile на 1 акаунті

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.2. Вигляд даних на формі надісланих повідомлень на 1 акаунті

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.3. Вигляд даних на формі Profile на 2 акаунті

Изображение выглядит как облако, небо, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.4. Вигляд даних на формі надісланих повідомлень на 2 акаунті

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.5. Вигляд змісту файлу-зберігання даних users.json

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.6. Вигляд змісту файлу-зберігання даних messages.json

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6.7. Вигляд файлів у Провіднику Windows

## **Алгоритм побудови кодів**

1. **Створити універсальний сервіс збереження**  
   – Додати клас FileStore<T> у Services, який при ініціалізації створює файл у App\_Data та має методи Load()/Save().
2. **Налаштувати залежності в Program.cs**  
   – Зареєструвати FileStore<User> і FileStore<Message> як singleton-сервіси, передавши їм IWebHostEnvironment та імена файлів (users.json, messages.json).  
   – Підключити MVC (AddControllersWithViews()), сесію (AddSession()) та доступ до HTTP-контексту (AddHttpContextAccessor()).  
   – Встановити маршрут за замовчуванням на Splash/Index.
3. **Реалізувати AccountController**  
   – Для реєстрації, входу, виходу, перегляду профілю та редагування полів завантажувати список користувачів із \_users.Load(), вносити зміни, а потім зберігати через \_users.Save().
4. **Реалізувати MessagesController**  
   – У дії Send (GET) подавати список імен із \_users.Load().  
   – В Send (POST) додавати новий Message з Id = Guid.NewGuid() та SentAt = DateTime.Now у список із \_msgs.Load(), після чого викликати \_msgs.Save().  
   – У Inbox, Sent та Details — фільтрувати та віддавати повідомлення з \_msgs.Load().
5. **Створити Views**  
   – Використати ті ж Razor-шаблони для реєстрації, входу, профілю, редагування та повідомлень (Send, Inbox, Sent, Details).
6. **Налаштувати навігацію**  
   – В \_Layout.cshtml через ctrl і act побудувати меню відповідно до схеми: Splash → Login ↔ Register → Profile ↔ Send/Inbox/Sent/Details.
7. **Тестування**  
   – Перевірити всі сценарії: реєстрація → вхід → редагування профілю → відправка та перегляд повідомлень → вихід.

## **Функціональність**

* **Повне файлове зберігання** користувачів і повідомлень у users.json та messages.json.
* **Реєстрація/авторизація** без СУБД через FileStore<User>.
* **Профіль** із можливістю змінювати ім’я, email, телефон та пароль.
* **Відправка повідомлень** із формою вибору отримувача, темою та текстом.
* **Перегляд Inbox/Sent** із відсортованими списками повідомлень.
* **Деталі повідомлення** з перевіркою прав доступу (лише відправник або отримувач).

## **Висновок**

Перехід на файлове зберігання даних через уніфікований FileStore<T> дозволив реалізувати систему без СУБД, зберігаючи всю інформацію в текстових JSON-файлах. Такий підхід спрощує розгортання, робить проєкт легким для перенесення та подальшого масштабування, водночас зберігаючи повну функціональність реєстрації, авторизації, управління профілем та обміну повідомленнями.

# **Загальний Висновок**

У результаті виконання Лабораторної роботи №9 було створено повноцінний MVC WEB-застосунок із широким спектром функціональних можливостей на базі ASP.NET. Зокрема, реалізовано реєстрацію, авторизацію, профіль користувача з редагуванням даних, обмін повідомленнями між користувачами та зручний інтерфейс перегляду надісланих та вхідних листів. Всі компоненти системи були об’єднані у єдиний логічний сценарій взаємодії відповідно до завдання.

Під час виконання було досягнуто таких результатів:

* Створено профіль користувача з можливістю змінювати параметри (ім’я, email, телефон, пароль) на окремих сторінках.
* Реалізовано обмін повідомленнями між користувачами із збереженням даних у JSON-файлах.
* Впроваджено екран заставки (Splash Screen), що формує перше враження про систему.
* Забезпечено клієнтську та серверну валідацію для підвищення надійності введення даних.
* Впроваджено маршрутизацію, яка точно відповідає заданій схемі переходів між сторінками.
* Реалізовано повністю файлову архітектуру зберігання даних (без використання БД) у Task\_6 — що демонструє гнучкість та адаптивність підходу.

Отже, поставлені цілі та завдання було виконано в повному обсязі. Отримані навички дозволяють глибше зрозуміти архітектуру MVC, використання сесій, форм валідації, роботу з файлами у форматі JSON, а також структурування взаємодії між різними частинами веб-застосунку. Сформована система є основою для подальшого розширення та вдосконалення.