

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ

Кафедра кібербезпеки та захисту інформації

Методичні вказівки до практичних робіт
з освітнього компоненту (дисципліни)
КІБЕРБЕЗПЕКА СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
для здобувачів спеціальності F5 – Кібербезпека та захист інформації
галузі знань F – Інформаційні технології
другого (магістерського) рівня освіти
для денної та заочної форми навчання
Частина 1

ОДЕСА – 2025

Методичні вказівки до практичних робіт з освітнього компоненту **«Кібербезпека сучасних інформаційних технологій»** частина 1 для здобувачів спеціальності F5 – Кібербезпека та захист інформації галузі знань F – Інформаційні технології другого (магістерського) рівня освіти для денної та заочної форми навчання / Укл.: Лебедєва О.Ю. – Одеса, 2025. – 52 с.

Розробник: к.т.н., доцент Лебедєва Олена Юріївна, доцент кафедри кібербезпеки та захисту інформації.

Схвалено на засіданні кафедри «Кібербезпека та захист інформації» Протокол від №2 від 06.10.2025.

Схвалено на засіданні НМК ННІ ІТІП Протокол від №2 від 14.10.2025

Зміст

Вступ.....	4
Практична робота 1. Створення веб-сайту.....	5
1. Теоретичний матеріал	5
2. Завдання до практичної роботи 1.....	12
Практична робота 2. Створення веб-проекту за технологією ASP.NET Web Forms. JSON-файли.....	13
1. Теоретичний матеріал	13
2. Завдання до практичної роботи 2.....	22
Практична робота 3. Створення веб-проекту за технологією Asp.Net Core MVC.....	23
1. Теоретичний матеріал	23
2. Завдання до практичної роботи 3.....	35
Практична робота 4. Створення нових сторінок у веб-проекті	36
1. Теоретичний матеріал	36
2. Завдання до практичної роботи 4.....	44
Практична робота 5. Додавання на сторінку форми	45
1. Теоретичний матеріал	45
2. Завдання до практичної роботи 5.....	51
Література.....	52

Вступ

Освітній компонент (навчальна дисципліна) «Кібербезпека сучасних інформаційних технологій» орієнтований на засвоєння здобувачами вищої освіти другого (магістерського) рівня методів розробки та захисту веб-додатків від найпоширеніших кіберзагроз. Під час опанування освітнього компонента (навчальної дисципліни) здобувачі набуватимуть теоретичних та практичних навичок дослідження, розробки та впровадження заходів захисту веб-додатків.

Метою освітнього компонента (навчальної дисципліни) є забезпечення розвитку фахових компетентностей майбутніх магістрів шляхом оволодіння сучасними підходами до розробки та впровадження заходів захисту веб-додатків від найпоширеніших кіберзагроз.

Завданнями, що мають бути вирішені у процесі викладання освітнього компонента (навчальної дисципліни), є набуття здобувачами компетентностей за освітньою програмою.

Інтегральна компетентність.

Здатність особи розв'язувати задачі дослідницького та/або інноваційного характеру у сфері інформаційної безпеки та/або кібербезпеки.

Загальні компетентності (КЗ)

КЗ-1. Здатність застосовувати знання у практичних ситуаціях

Фахові (спеціальні) компетентності (ФК/СК):

КФ1. Здатність обґрунтовано застосовувати, інтегрувати, розробляти та удосконалювати сучасні інформаційні технології, фізичні та математичні моделі, а також технології створення та використання прикладного і спеціалізованого програмного забезпечення для вирішення професійних задач у сфері інформаційної безпеки та/або кібербезпеки.

КФ7. Здатність досліджувати, розробляти та впроваджувати методи і заходи протидії кіберінцидентам, здійснювати процедури управління, контролю та розслідування, а також надавати рекомендації щодо попередження та аналізу кіберінцидентів в цілому.

Вивчення освітнього компонента (навчальної дисципліни) передбачає здобуття таких *результатів навчання (РН):*

РН4. Застосовувати, інтегрувати, розробляти, впроваджувати та удосконалювати сучасні інформаційні технології, фізичні та математичні методи і моделі в сфері інформаційної безпеки та/або кібербезпеки.

РН8. Досліджувати, розробляти і супроводжувати системи та засоби інформаційної безпеки та/або кібербезпеки на об'єктах інформаційної діяльності та критичної інфраструктури.

Практична робота 1. Створення веб-сайту

Мета роботи: ознайомитися зі створенням веб-сайтів за допомогою HTML та CSS. На прикладі веб-сайту набути практичних навичок побудови веб-сайтів за допомогою HTML та CSS.

1. Теоретичний матеріал

HTML (від англ. HyperText Markup Language – мова гіпертекстової розмітки) – стандартизована мова розмітки документів у Всесвітній павутині. Більшість вебсторінок мають зміст розмітки мовою HTML (або XHTML).

Мова HTML інтерпретується браузером; отриманий в результаті інтерпретації форматований текст відображається на екрані монітора комп'ютера або мобільного пристрою.

<!DOCTYPE html> - оголошення типу документа.

Елемент <html> містить весь вміст на всій сторінці, і іноді його називають "кореневий елемент".

Елемент <head> є контейнером для метаданих (даних про дані) і розміщується між тегом <html> та тегом <body>. Метадані – це дані про документ HTML. Метадані не відображаються. Метадані зазвичай визначають назву документа, набір символів, стилі, скрипти та іншу метаінформацію.

Наведені нижче елементи можуть входити в елемент <head>:

- <title> (обов'язковий в кожному HTML документі)
- <style>
- <base>
- <link>
- <meta>
- <script>
- <noscript>

Тег <body> визначає тіло документа. Елемент <body> містить увесь вміст документа HTML, наприклад заголовки, абзаци, зображення, гіперпосилання, таблиці, списки тощо. В HTML документі може бути лише один елемент <body>.

Теги (дескриптори) у HTML – це спеціальні елементи розмітки, що визначають, як даний документ HTML має інтерпретуватися програмою перегляду інформації – браузером. Кожний тег описує посилання на якийсь об'єкт з об'єктної моделі браузера.

Теги HTML бувають одно- і двокомандні.

Однокомандні теги зазвичай використовуються для зміни вигляду виведеної на екран інформації. Вони задаються в такому вигляді:

<команда [атрибут=значення [атрибут=значення]...]>текст,

де команда – команда HTML;

атрибут=значення – задання конкретних значень параметрам (атрибутам) команди (не є обов'язковими);

текст – текст, що виводиться на екран тільки для читання у вигляді, обумовленому командою та її атрибутами

Двокомандні теги мають такий синтаксис:

*<команда [атрибут=значення [атрибут=значення]...]>
[текст][тег][[текст][тег]...]
</команда [атрибут=значення [атрибут=значення]...]>*

де / – використовується у випадку, коли тег є останнім (закриваючим) у блоці тегів.

HTML - елемент = тег, що відкриває + тег, що закриває, + вміст

Основними частинами елемента є:

- Відкриваючий тег: Він складається з назви (позначення) елемента, поміщеного всередині кутових дужок. Цей тег є ознакою початку елемента, з цього моменту тег починає впливати на наступний після нього вміст.
- Тег, що закриває: виглядає як і відкриває, але містить сліш перед назвою тега. Він є ознакою кінця елемента.
- Вміст: вмістом є текст, який відображається на сторінці.

Не всі елементи відповідають вищезгаданому шаблону: тег, що відкриває, контент, тег, що закриває. Деякі елементи складаються з одного тега і зазвичай використовуються для вставки чогось у місце документа, де розміщені.

Елементи також можуть мати атрибути. Атрибути містять додаткову інформацію щодо елемента.

Атрибут повинен мати:

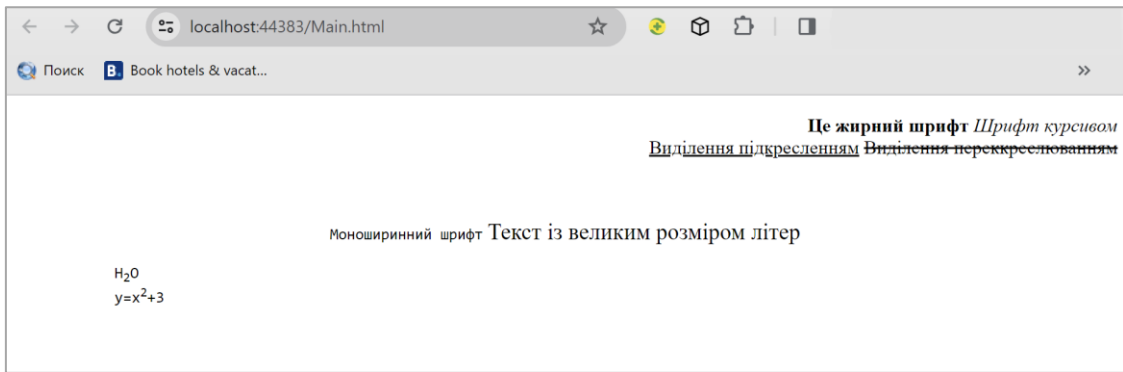
- Пробіл між атрибутом та ім'ям елемента (або попереднім атрибутом, якщо елемент вже має один або кілька атрибутів).
- Ім'я атрибута та наступний за ним знак рівності.
- Значення атрибута, укладене у лапки.

Існують атрибути, написані без значення. Такі атрибути називаються булеві, і вони можуть мати лише одне значення, яке збігається з його ім'ям. Для стислості можна записувати лише ім'я атрибута без його значення.

Розглянемо декілька прикладів застосування тегів.

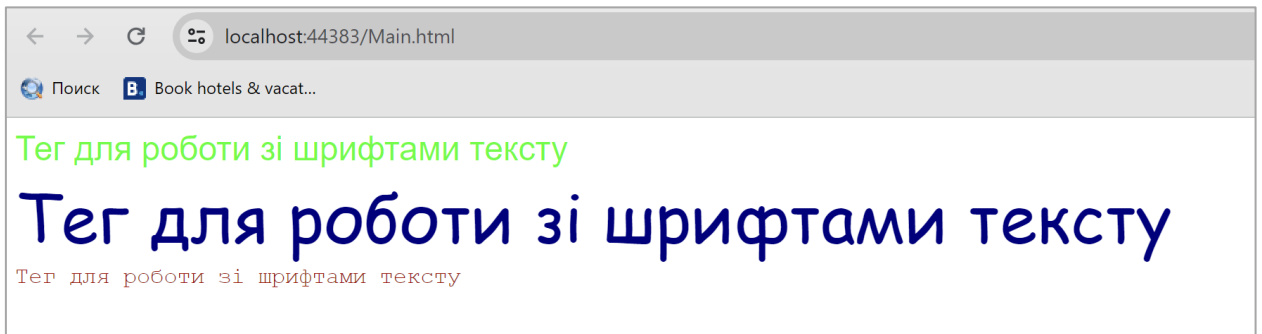
Приклад з тегами фізичного форматування тексту:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Теги HTML</title>
</head>
<body>
  <p align="right">
    <b>Це жирний шрифт</b>
    <i>Шрифт курсивом</i><br />
    <u>Виділення підкресленням</u>
    <strike>Виділення перекреслюванням</strike>
  </p>
  <br />
  <p align="center">
    <tt>Моноширинний шрифт</tt>
    <big>Текст із великим розміром літер</big><br />
    <pre>
      H<sub>2</sub>O
      y=x<sup>2</sup>+3
    </pre>
  </p>
</body>
</html>
```



Приклад з налаштування шрифту:

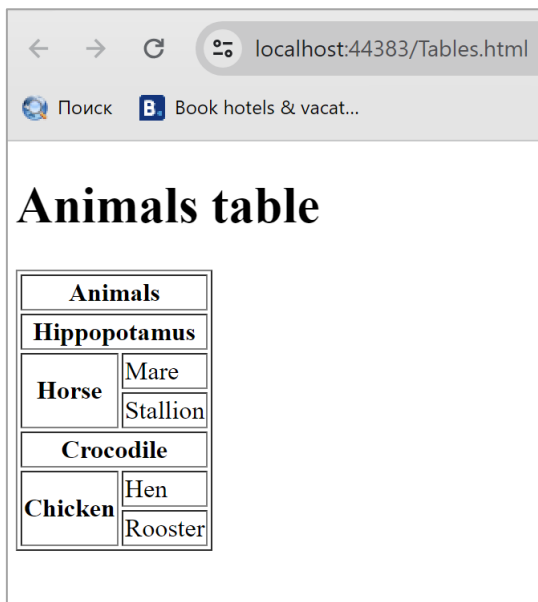
```
<body>
  <font face="Arial" size="5" color="Lime">Тег для роботи зі шрифтами тексту</font><br>
  <font face="Comic Sans MS" size="7" color="Navy">Тег для роботи зі шрифтами тексту</font><br>
  <font face="Courier New" size="3" color="#AA0000">Тег для роботи зі шрифтами тексту</font>
</body>
```



Приклад зі створення таблиці:

```
<h1>Animals table</h1>

<table border="1">
  <tr>
    <th colspan="2">Animals</th>
  </tr>
  <tr>
    <th colspan="2">Hippopotamus</th>
  </tr>
  <tr>
    <th rowspan="2">Horse</th>
    <td>Mare</td>
  </tr>
  <tr>
    <td>Stallion</td>
  </tr>
  <tr>
    <th colspan="2">Crocodile</th>
  </tr>
  <tr>
    <th rowspan="2">Chicken</th>
    <td>Hen</td>
  </tr>
  <tr>
    <td>Rooster</td>
  </tr>
</table>
```



Приклад з використанням форм:

```
<body bgcolor="Yellow">
  <h2>Реєстрація</h2>

  <form method="post">
    <div>
      <label for="name">Ім'я</label>
      <input type="text" name="name" required>
    </div><br>
    <div class="radio">
      <span>Стать</span><br>
      <label>
        <input type="radio" name="sex" value="male">чоловічий
        <div class="radio-control male"></div>
      </label>
      <label>
        <input type="radio" name="sex" value="female">жіночий
        <div class="radio-control female"></div>
      </label>
    </div><br>
    <div>
      <label for="email">E-mail</label>
      <input type="email" name="email" required>
    </div><br>
    <div>
      <label for="country">Виберіть країну проживання</label><br>
      <select name="country">
        <option value="Ukrain">Україна</option>
        <option value="Poland">Польща</option>
        <option value="Germany">Німеччина</option>
      </select>
      <div class="select-arrow"></div>
    </div><br>
    <button type="submit">Відправити</button>
  </form>
</body>
```

localhost:44383/Main.html

Поиск Book hotels & vacat...

Реєстрація

Ім'я

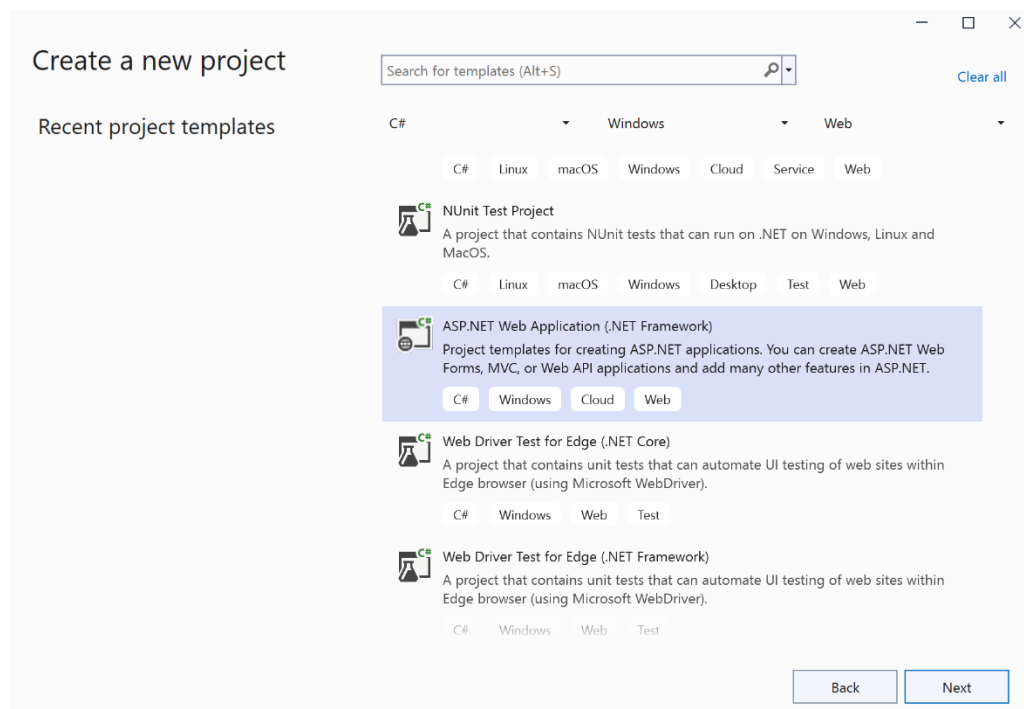
Стать
☐ чоловічий
☐ жіночий

E-mail

Виберіть країну проживання
Україна ▾

Відправити

В якості середовища для створення веб-сторінки Visual Studio Community.
При створенні обираємо тип проекту ASP.NET Web Application (.Net Framework).



На наступному кроці вказується місцезнаходження проекту та його назва. Натискаємо кнопку Create. Далі пропонується вибрати чим буде начинене поточний проект. Обираємо Empty.

Create a new ASP.NET Web Application

**Empty**
An empty project template for creating ASP.NET applications. This template does not have any content in it.

**Web Forms**
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

**MVC**
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

**Web API**
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

**Single Page Application**
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication

None

Add folders & core references

☐ Web Forms

☐ MVC

☐ Web API

Advanced

☒ Configure for HTTPS

☐ Docker support
(Requires [Docker Desktop](#))

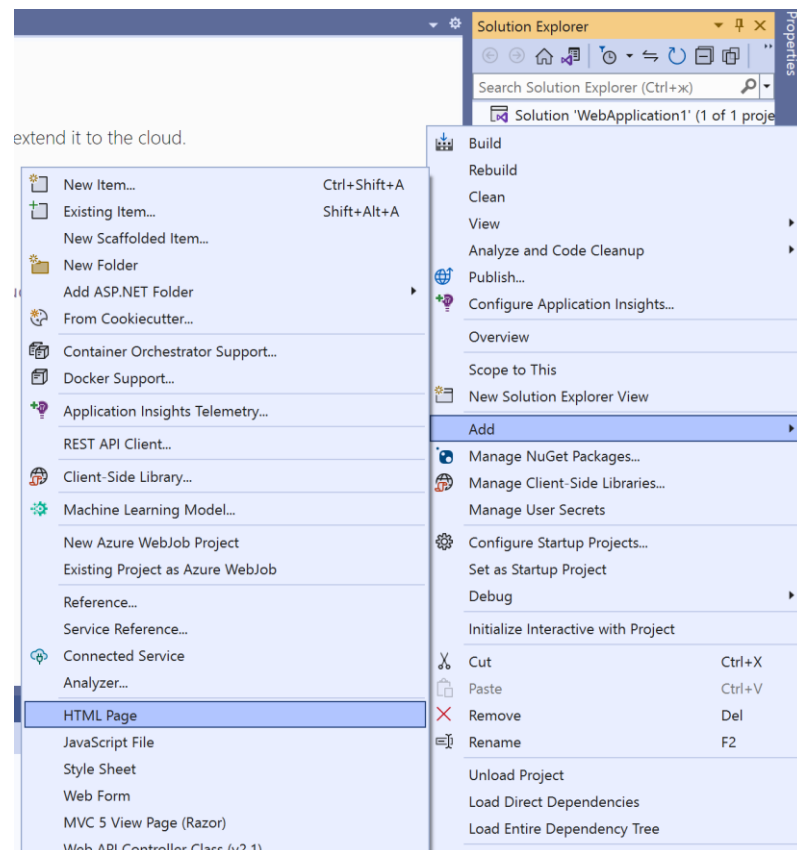
☐ Also create a project for unit tests

WebApplication1.Tests

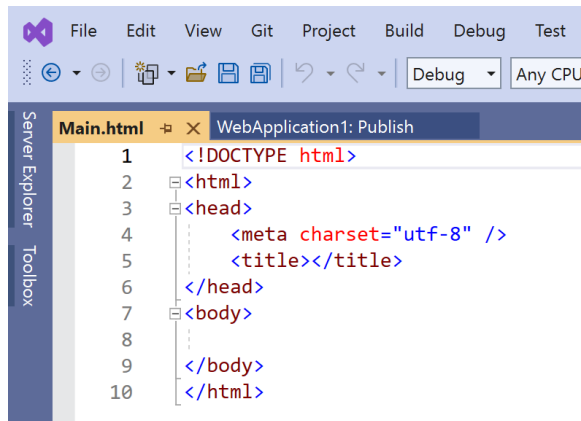
Back

Create

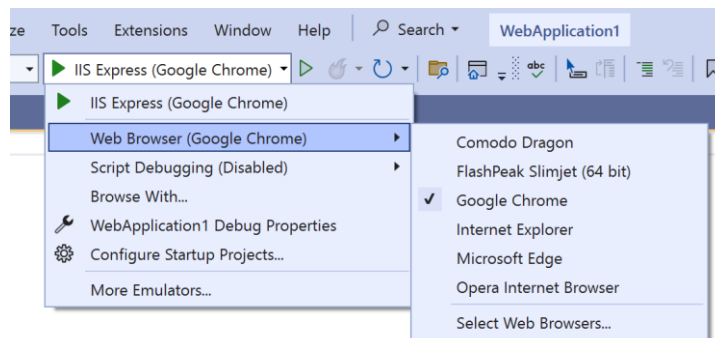
Створюється пустий проект. Далі до цього проекту потрібно додати HTML Page та вказати її ім'я.



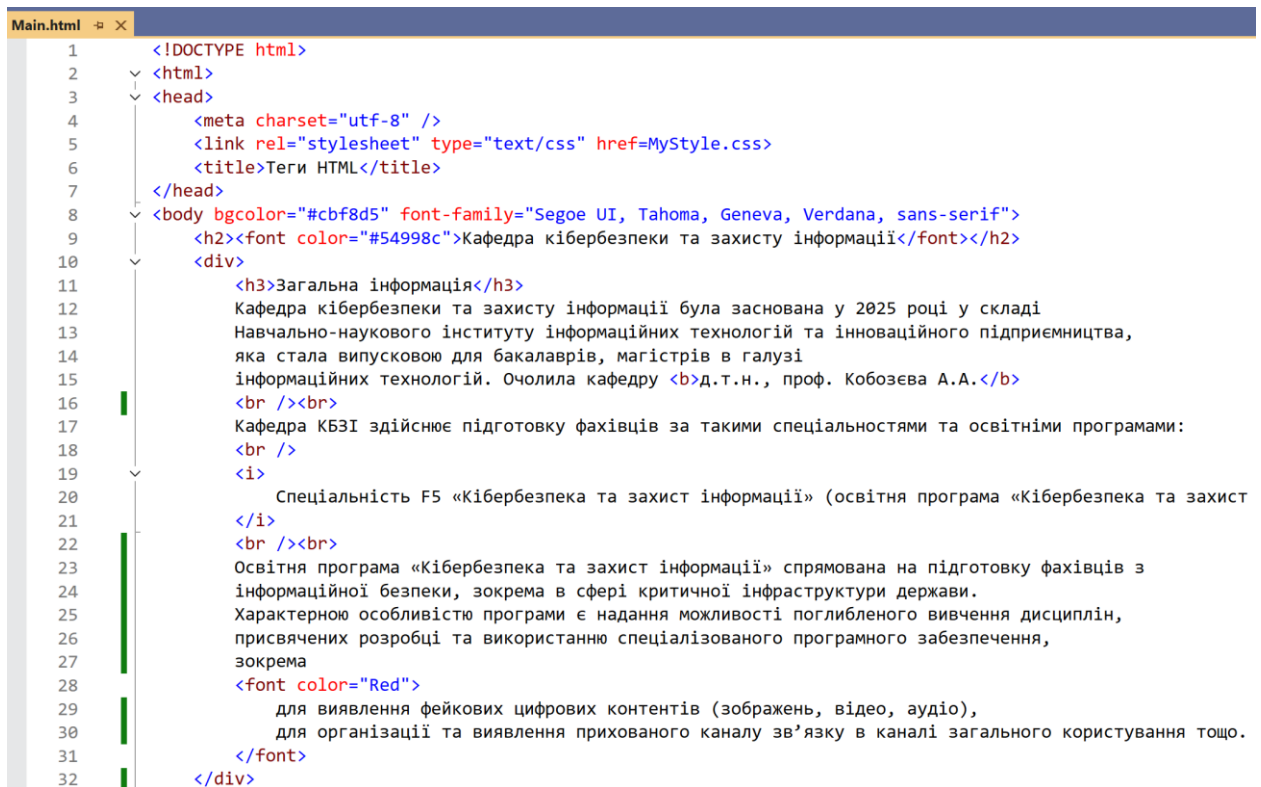
В результаті в робочій області середовища з'явиться наступна розмітка



Для запуску обираємо браузер та запускаємо проект на виконання.



Наприклад, створимо веб сторінку про нашу кафедру:



```

33 <br>
34 <div>
35 <h3>Склад кафедри</h3>
36 <table style="border-collapse: collapse; border: none;">
37 <tr>
38 <td><img src=Resources/Кобозева.png alt="Зображення 1" width="100"></td>
39 <td style="padding: 10px;">Кобозева А.А. <br>завідувач кафедри, професор, доктор технічних
40 <td><img src=Resources/Лебедева.png alt="Зображення 2" width="100"></td>
41 <td style="padding: 10px;">Лебедева О.Ю. <br>доцент, кандидат технічних наук</td>
42 </tr>
43 <tr>
44 <td><img src=Resources/Зоріло.png alt="Зображення 3" width="100"></td>
45 <td style="padding: 10px;">Зоріло В.В. <br>доцент, кандидат технічних наук</td>
46 <td><img src=Resources/Трифоновна.png alt="Зображення 4" width="100"></td>
47 <td style="padding: 10px;">Трифоновна К.О. <br>старший викладач</td>
48 </tr>
49 <tr>
50 <td><img src=Resources/Сокальський.png alt="Зображення 5" width="100"></td>
51 <td>Сокальський С.М.<br>асистент</td>
52 </tr>
53 </table>
54 </div>
55 </body>
56 </html>

```

Результат:

Кафедра кібербезпеки та захисту інформації

Загальна інформація

Кафедра кібербезпеки та захисту інформації була заснована у 2025 році у складі Навчально-наукового інституту інформаційних технологій та інноваційного підприємництва, яка стала випусковою для бакалаврів, магістрів в галузі інформаційних технологій. Очолила кафедру д.т.н., **проф. Кобозєва А.А.**

Кафедра КБЗІ здійснює підготовку фахівців за такими спеціальностями та освітніми програмами:
Спеціальність F5 «Кібербезпека та захист інформації» (освітня програма «Кібербезпека та захист інформації»)

Освітня програма «Кібербезпека та захист інформації» спрямована на підготовку фахівців з інформаційної безпеки, зокрема в сфері критичної інфраструктури держави. Характерною особливістю програми є надання можливості поглибленого вивчення дисциплін, присвячених розробці та використанню спеціалізованого програмного забезпечення, зокрема для виявлення **фейкових цифрових контентів (зображень, відео, аудіо)**, для організації та виявлення прихованого каналу зв'язку в каналі загального користування тощо.

Склад кафедри



Кобозєва А.А.
завідувач кафедри, професор, доктор технічних наук



Лебедева О.Ю.
доцент, кандидат технічних наук



Зоріло В.В.
доцент, кандидат технічних наук



Трифоновна К.О.
старший викладач

2. Завдання до практичної роботи 1

Оберіть тематику для веб-сайту. Цю тему ви будете використовувати протягом всіх лабораторних занять. Створіть новий веб-сайт за допомогою HTML та CSS. Розробіть сайт з п'яти сторінок. Додайте до сторінок відповідний текст, рисунки, перелічення та таблиці. Створіть сторінку з формою. За бажанням можна використовувати фрейми.

Практична робота 2. Створення веб-проекту за технологією ASP.NET Web Forms. JSON-файли

Мета роботи: ознайомитися з технологією ASP.NET Web Forms для створення веб-додатків та збереження даних в JSON-файлах. Набути практичних навичок в створенні веб-додатків за допомогою технології ASP.NET Web Forms та використанню JSON-файлів.

1. Теоретичний матеріал

JSON, скорочення від JavaScript Object Notation, є популярним форматом обміну даними, який широко застосовується у веб-розробці. Він являє собою формат, що легко читається, заснований на JavaScript об'єктах і масивах.

JSON-файли використовуються для передачі даних між клієнтом та сервером. Вони також широко використовуються в API, щоб обмінюватися інформацією між різними програмами.

JSON-файл складається з пар "ключ-значення", де ключі – це рядки, а значення можуть бути будь-якого типу даних: рядки, числа, логічні значення, об'єкти або масиви. Це робить JSON універсальним та зручним для використання у різних сценаріях.

Формат заснований на двох структурах даних: об'єктах та масивах. Об'єкти в JSON є неупорядкованими колекціями пар «ключ-значення». Ключі мають бути рядками, а значення можуть бути будь-якого типу даних. Масиви в JSON є упорядкованими списками значень. Значення можуть бути будь-якого типу даних.

Приклад JSON-файлу:

```
{
  "name": "John",
  "age": 30,
  "city": "New York",
  "hobbies": ["reading", "traveling", "photography"],
  "isStudent": false
}
```

У цьому прикладі ми маємо об'єкт з п'ятьма полями: "name", "age", "city", "hobbies" та "isStudent", "name" і "city" є рядками, "age" – числом, "hobbies" – масивом рядків, а "isStudent" – логічним значенням.

Простір імен System.Text.Json надає функціональні можливості для серіалізації в нотацію об'єктів JavaScript (JSON) та десеріалізації об'єктів із цієї нотації. Серіалізація – це процес перетворення стану об'єкта, тобто значень його властивостей у форму, яка може зберігатися або передаватися. Серіалізована форма не містить жодних відомостей про пов'язані методи об'єкта. Десеріалізація відновлює об'єкт із серіалізованої форми.

Дизайн System.Text.Json бібліотеки підкреслює високу продуктивність та низьке виділення пам'яті за великим набором функцій. Вбудована підтримка UTF-8 оптимізує процес читання та запису тексту JSON у кодуванні UTF-8, яке є найпоширенішим кодуванням для даних в Інтернеті та файлів на диску.

Для збереження об'єкта в json у класі JsonSerializer визначено статичний метод Serialize() та його асинхронний двійник SerializeAsync(), які мають низку перевантажених версій.

Для десеріалізації коду json в об'єкт C# застосовується метод Deserialize() та його асинхронний двійник DeserializeAsync(), які мають різні версії.

Розглянемо застосування класу простому прикладі. Серіалізуємо та десеріалізуємо найпростіший об'єкт:

```

using System.Text.Json;

Person tom = new Person("Tom", 37);
string json = JsonSerializer.Serialize(tom);
Console.WriteLine(json);
Person? restoredPerson = JsonSerializer.Deserialize<Person>(json);
Console.WriteLine(restoredPerson?.Name); // Tom

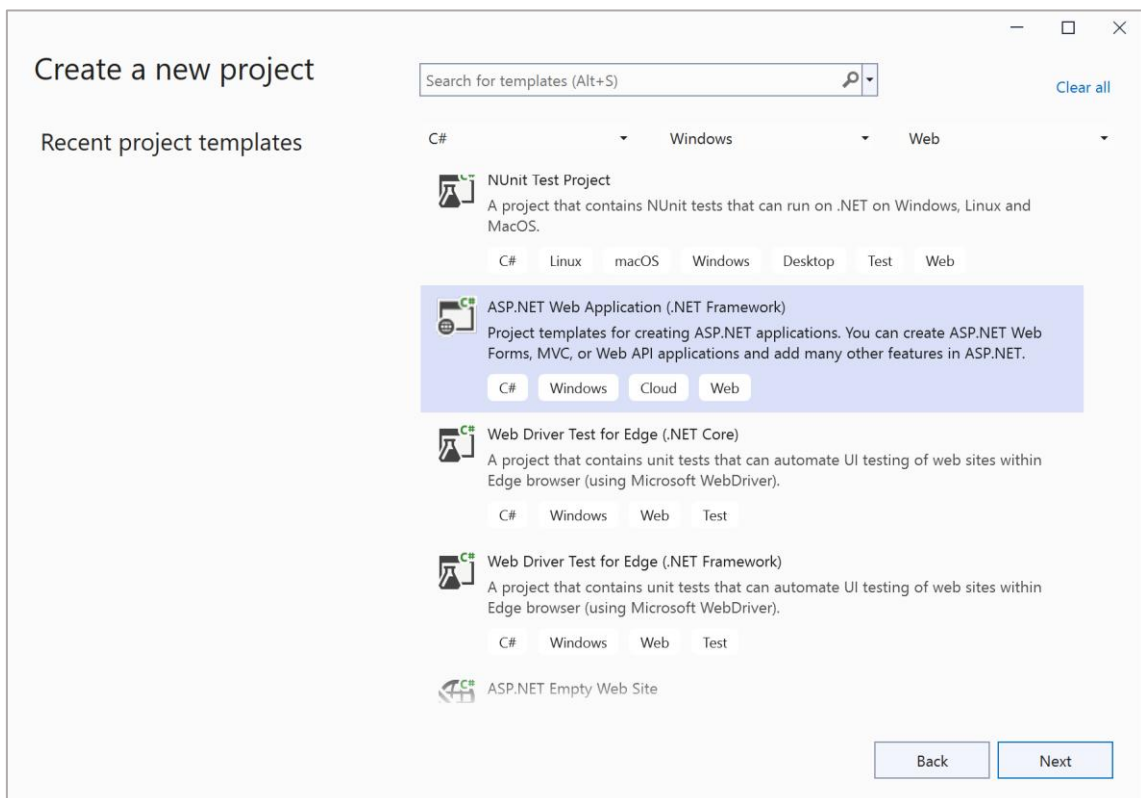
class Person
{
    public string Name { get; }
    public int Age { get; set; }
    public Person(string name, int age)
    {
        Name = name;
        Age = age;
    }
}

```

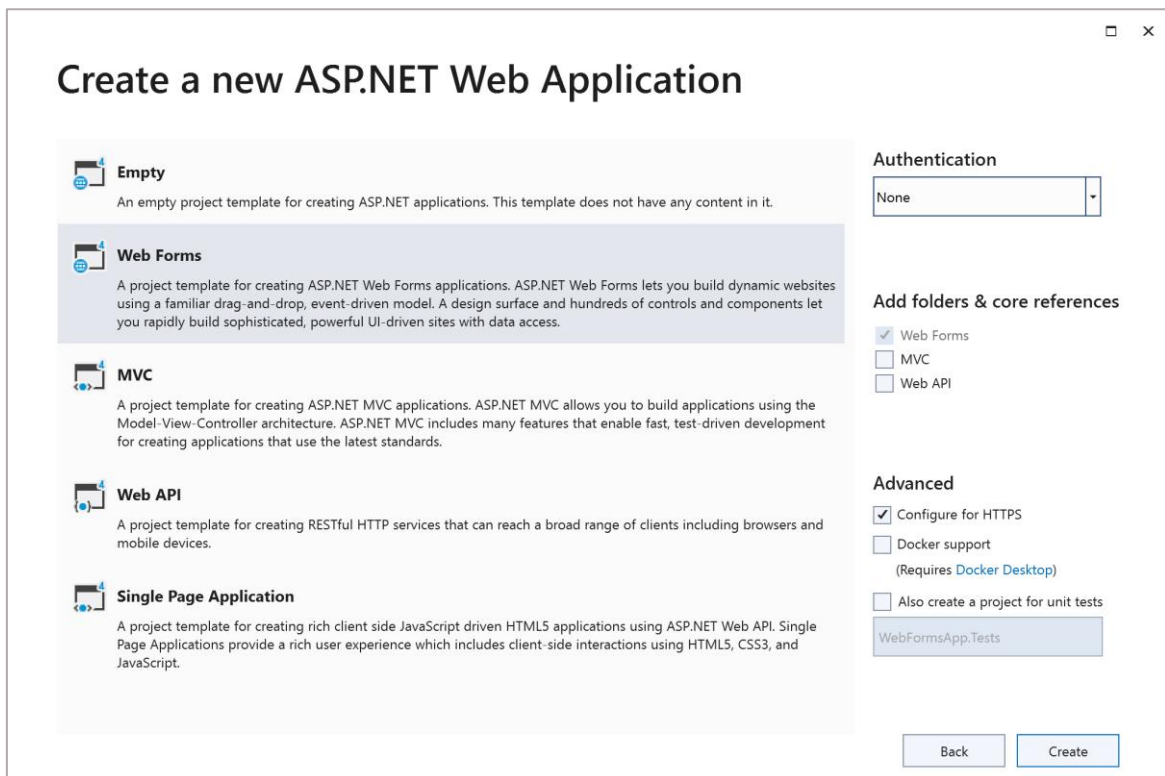
Тут спочатку серіалізуємо за допомогою методу `JsonSerializer.Serialize()` об'єкт типу `Person` у рядок з кодом `json`. Потім обернемо з цього рядка об'єкт `Person` за допомогою методу `JsonSerializer.Deserialize()`.

Створимо сайт за допомогою ASP.NET Web Forms.

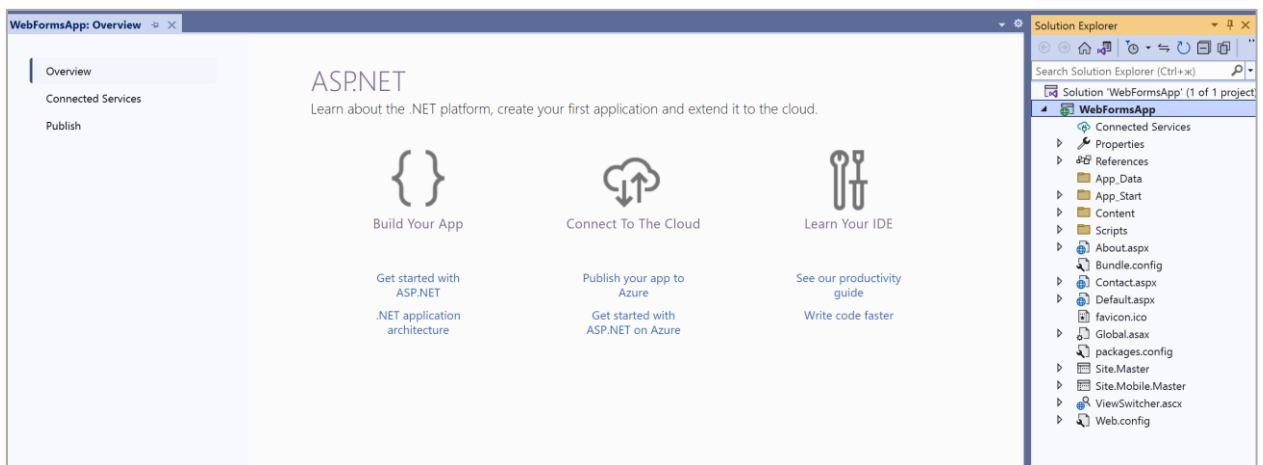
На першому кроці створимо проект ASP.NET:



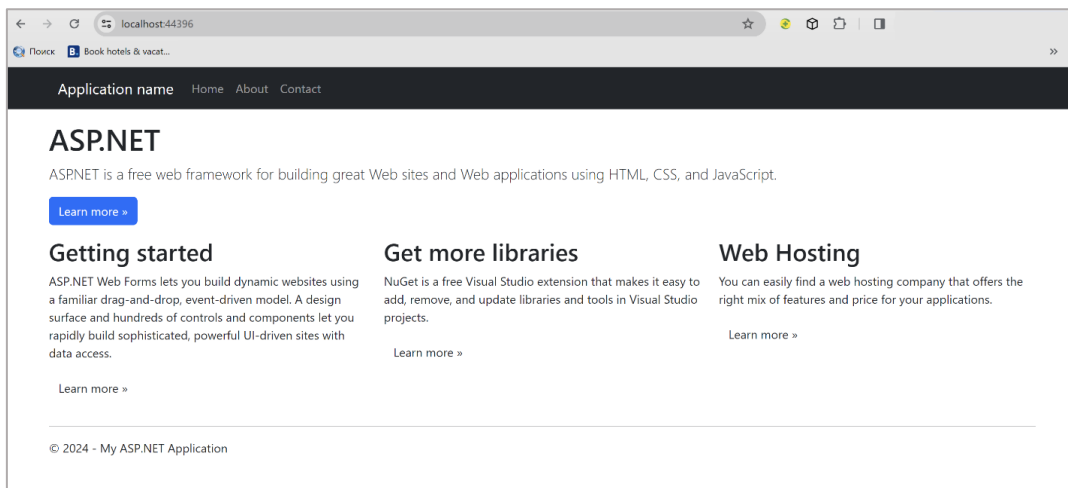
На наступному кроці вказуємо ім'я та місцезнаходження проекту. Далі вибираємо модель Web Forms:



Як тільки Visual Studio підготує шаблон проекту, його можна буде запустити, щоб перевірити. Для цього натисніть кнопку IIS Express:



Visual Studio скомпілює всі файли, запустить сайт і відкриє його у браузері за замовчанням.



У створеному шаблоні є кілька сторінок та стилів. Кожна сторінка складається з трьох файлів:

- Page.aspx – файл із HTML-кодом сторінки.
- Page.aspx.cs – клас, який відповідає за логіку сторінки.
- Page.aspx.designer.cs – клас, який є проширком між HTML і C#.

```
<%@ Page Title="About" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBe
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
  <main aria-labelledby="title">
    <h2 id="title"><%= Title %>.</h2>
    <h3>Your application description page.</h3>
    <p>Use this area to provide additional information.</p>
  </main>
</asp:Content>
```

Рисунок 2.1 – Приклад файлу About.aspx

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebFormsApp
{
    public partial class About : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}
```

Рисунок 2.2 – Приклад файлу About.aspx.cs

About.aspx містить лише фрагмент готової сторінки. Решта міститься у файлі Site.Master або Site.Mobile.Master. Контент із тега `<asp:Content>` (About.aspx) вбудовується на місце тегу `<asp:ContentPlaceHolder>` (Site.Master) з ідентичним ID.

Файл .aspx зазвичай починається з набору директив. Директиви ASP.NET завжди позначаються маркерами `<%@...%>` і можуть містити різноманітні атрибути, які інформують середовище ASP.NET про те, як слід обробляти конкретну директиву.

На початку файлу About.aspx знаходиться наступний елемент:

```
<%@ Page Title="About" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true"
CodeBehind="About.aspx.cs" Inherits="WebFormsApp.About" %>
```

Це називається сторінковою директивою (чи директивою Page). Визначені атрибути надають ASP.NET деталі про файл веб-форми. Наприклад, атрибут CodeBehind повідомляє ASP.NET, який файл класу C# містить код, асоційований із веб-формою. У цьому прикладі це файл About.aspx.cs, який є файлом відокремленого коду для About.aspx.

Розглянемо приклад: додаємо до файлу About.aspx властивість, яку ініціалізуємо в файлі About.aspx.cs.

Код до About.aspx

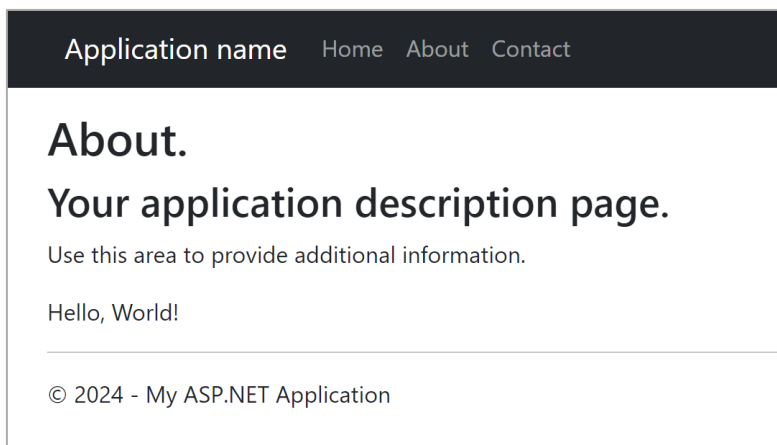
```
<main aria-labelledby="title">
  <h2 id="title"><%: Title %>.</h2>
  <h3>Your application description page.</h3>
  <p>Use this area to provide additional information.</p>

  <p><%: Text %></p><!--Property output-->
</main>
```

Код до About.aspx.cs

```
public string Text;
protected void Page_Load(object sender, EventArgs e)
{
    Text = "Hello, World!"; //When loading, the value of the Text property is set
}
```

Тепер при запуску проекту на сторінці About з'явиться рядок «Hello, World!»



Щоб створити інші сторінки, натисніть правою кнопкою назву проекту і в контекстному меню виберіть Add —> Web Form. Буде створена сторінка з таким кодом:

```

WebFormContact.aspx About.aspx About.aspx.cs WebFormsApp: Overview
1 <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebFormContact.aspx.cs" Inherits="WebFormsApp.WebFormContact" %>
2
3 <!DOCTYPE html>
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7 <title></title>
8 </head>
9 <body>
10 <form id="form1" runat="server">
11 <div>
12 </div>
13 </form>
14 </body>
15 </html>
16

```

У цій сторінці немає тегу `<asp:Content>`, тому шаблон із файлу `Site.Master` не відобразиться. Щоб змінити це, можна скопіювати код із `About.aspx`.

```

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
</asp:Content>

```

Додамо в меню сайту посилання на нову сторінку. Меню знаходиться у `Site.Master`. Додамо наступний рядок:

```

<li class="nav-item"><a class="nav-link" runat="server" href="~/WebFormContact">Contact form</a></li>

```

де `WebFormContact` – назва сторінки, яку ви створили.

Тоді код у `Site.Master` буде мати наступний вигляд:

```

<div class="collapse navbar-collapse d-sm-inline-flex justify-content-between">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item"><a class="nav-link" runat="server" href="~/>Home</a></li>
    <li class="nav-item"><a class="nav-link" runat="server" href="~/About">About</a></li>
    <li class="nav-item"><a class="nav-link" runat="server" href="~/Contact">Contact</a></li>
    <li class="nav-item"><a class="nav-link" runat="server" href="~/WebFormContact">Contact form</a></li>
  </ul>
</div>

```

Створимо на сторінці `WebFormContact` форму для заповнення даними користувача. Додаємо до файлу `WebFormContact.aspx` наступний код:

```

<%@ Page Title="Contact form" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="WebFormContact.aspx.cs" Inherits="WebFormsApp.WebFormContact" %>
<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
  <main aria-labelledby="title">
    <h3 id="title"><%: Title %>.</h3>
    <p>All fields must be filled in</p>

    <form method="post">
      <div>
        <label for="name">Name: </label>
        <input type="text" name="name" required />
      </div><br>
      <div>
        <label for="lastname">Last name: </label>
        <input type="text" name="lastname" required />
      </div><br>
      <div>
        <label for="email">E-mail: </label>
        <input type="text" name="email" required />
      </div><br>
    </form>
  </main>
</asp:Content>

```

```

        <div>
            <label for="phone">Phone: </label>
            <input type="text" name="phone" required />
        </div><br>
        <br>
        <asp:Button ID="ButtonSave" runat="server" Text="Save contact" OnClick="Button1_Click" />
    </form>
</main>
</asp:Content>

```

The screenshot shows a web browser window with the address bar displaying 'Поиск' and 'Book hotels & vacat...'. The page has a dark navigation bar with links: 'Application name', 'Home', 'About', 'Contact', and 'Contact form'. The main content area is titled 'Contact form.' and includes a note: 'All fields must be filled in'. Below this are four text input fields labeled 'Name:', 'Last name:', 'E-mail:', and 'Phone:'. At the bottom of the form is a 'Save contact' button. The footer of the page reads '© 2024 - My ASP.NET Application'.

Майже всі веб-додатки покладаються на будь-який різновид моделі даних, незалежно від технології, яка використовується для їх створення. Створимо просту модель даних. Кладніть правою кнопкою миші на назві проекту у вікні Solution Explorer і виберіть у контекстному меню пункт Add → Class. Створимо клас `ContactInfo`:

```

public class ContactInfo
{
    public string Name { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}

```

Для представлення відомостей про користувача будуть застосовуватись екземпляри класу `ContactInfo`. Створені об'єкти `ContactInfo` повинні записуватись у сховище. У реальному додатку в його якості зазвичай виступає база даних, нам потрібно щось швидке і просте, тому ми збираємося зберігати об'єкти в пам'яті. Перевага такого підходу пов'язана з простотою реалізації, проте слід враховувати, що дані будуть губитися при кожній зупинці або перезапуску програми.

Щоб визначити сховище, додайте в проект новий файл класу на ім'я `ContactRepository.cs` і помістіть код, показаний нижче:

```

public class ContactRepository
{
    private static ContactRepository repository = new ContactRepository();
    private List<ContactInfo> responses = new List<ContactInfo>();

    public static ContactRepository GetRepository()
    {
        return repository;
    }

    public IEnumerable<ContactInfo> GetAllResponses()
    {
        return responses;
    }

    public void AddResponse(ContactInfo response)
    {
        responses.Add(response);
    }
}

```

Сховище зазвичай має в своєму розпорядженні методи для створення, читання, оновлення та видалення об'єктів даних (разом званих методами CRUD (creating, reading, updating, deleting – створення, читання, оновлення, видалення), але в нашому випадку потрібна тільки можливість читання всіх об'єктів даних та додавання нових об'єктів даних.

На початку файлу WebFormContact.aspx знаходиться наступний елемент:

```

<%@ Page Title="Contact form" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="WebFormContact.aspx.cs"
Inherits="WebFormsApp.WebFormContact" %>

```

Це називається сторінковою директивою (чи директивою Page). Визначені атрибути надають ASP.NET деталі про файл веб-форми. Нас цікавить атрибут CodeBehind. Цей атрибут повідомляє ASP.NET, який файл класу C# містить код, асоційований з веб-формою. У нашому випадку це файл WebFormContact.aspx.cs, який є файлом відокремленого коду для WebFormContact.aspx.

Середовище Visual Studio групує разом пов'язані файли як одиночного елемента у вікні Solution Explorer, що спрощує навігацію за великими проектами. Якщо клацнути на стрілці зліва від запису WebFormContact.aspx, можна побачити файли, приховані середовищем Visual Studio. Одним з них є файл WebFormContact.aspx.cs, на який провадиться посилання за допомогою атрибуту CodeBehind.

Двічі клацніть файл WebFormContact.aspx.cs, щоб відкрити його в редакторі; з'явиться код, наведений нижче:

```

public partial class WebFormContact : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}

```

Базовим для класу відокремленого коду є клас System.Web.UI.Page, який містить кілька корисних методів та властивостей для відповіді на веб-запити. Нас цікавить метод Page_Load(), що викликається ASP.NET Framework при надходженні запитів для WebFormContact.aspx і надає можливість відреагувати такі запити.

Додаємо в цей клас метод `Button1_Click`, який буде обробляти натискання кнопки "Save contact". В цьому методі далі розташуємо код зчитування даних з форми та збереження їх у вигляді json-файлу.

```
protected void Button1_Click(object sender, EventArgs e)
{
    ContactInfo rsvp = new ContactInfo();

    if (TryUpdateModel(rsvp, new FormValueProvider(ModelBindingExecutionContext)))
    {
        ContactRepository.GetRepository().AddResponse(rsvp);
        string jsonString = System.Text.Json.JsonSerializer.Serialize(rsvp);

        var fileName = AppDomain.CurrentDomain.BaseDirectory + "ContactInfo.json";
        File.WriteAllText(fileName, jsonString);

        Response.Redirect("WebFormContactInfo.aspx");
    }
}
```

Метод `TryUpdateModel()` виконує процес, який називається прив'язкою моделі та передбачає використання значень даних із запиту браузера для заповнення властивостей відповідного об'єкта моделі даних. Ще одним аргументом методу `TryUpdateModel()` є об'єкт, який платформа ASP.NET повинна застосовувати для отримання необхідних значень - ми використовуємо клас `System.Web.Mvc.FormValueProvider`, який надає значення даних форми. В результаті виклику методу `TryUpdateModel()` властивості об'єкта `ContactInfo` оновлюються, щоб відобразити значення даних, які користувач надіслав усередині форми. Потім об'єкт `ContactInfo` розміщується в сховищі.

Користувачеві необхідно надати будь-який відгук після того, як він відправив форму, і це робиться за допомогою методу `Response.Redirect()`, який виконує перенаправлення браузера користувача. В нашому випадку перенаправляємо на сторінку `WebFormContactInfo.aspx`.

Створимо нову сторінку `WebFormContactInfo.aspx` та до неї додаємо, наприклад, такий код:

```
<%@ Page Title="Contact information" Language="C#" AutoEventWireup="true" MasterPageFile="~/Site.Master" CodeBehind="WebFormContactInfo.aspx" Inherits="WebFormContactInfo" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
    <main aria-labelledby="title">
        <h3 id="title"><%: Title %></h3>
        <div>
            Dear <%: Name %> <%: LastName %>, your email: <%: Email %> and phone number: <%: Phone %> have been saved.
            We will contact you shortly.
        </div>
    </main>
</asp:Content>
```

А до класу `WebFormContactInfo.aspx.cs` додаємо код зчитування даних з json – файлу.

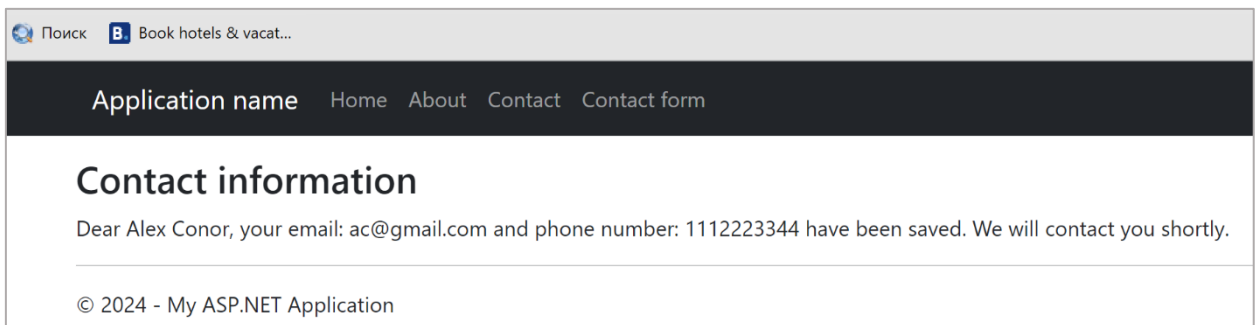
```

public partial class WebFormContactInfo : System.Web.UI.Page
{
    public string Name, LastName, Email, Phone;

    protected void Page_Load(object sender, EventArgs e)
    {
        var fileName = AppDomain.CurrentDomain.BaseDirectory + "ContactInfo.json";
        string jsonString = File.ReadAllText(fileName);
        ContactInfo rsvp = System.Text.Json.JsonSerializer.Deserialize<ContactInfo>(jsonString);

        Name = rsvp.Name;
        LastName = rsvp.LastName;
        Email = rsvp.Email;
        Phone = rsvp.Phone;
    }
}

```



2. Завдання до практичної роботи 2

Створіть новий проект ASP.NET Web Forms. Модифікуйте існуючі сторінки та додайте нові, щоб повторити сайт який ви розробляли на першій лабораторній роботі. Додайте можливість роботи з файлами json.

Практична робота 3. Створення веб-проекту за технологією Asp.Net Core MVC

Мета роботи: ознайомитися зі створенням веб-проекту за технологією Asp.Net Core MVC. На прикладі простого веб-застосування набути практичних навичок побудови веб-проекту за технологією Asp.Net Core MVC.

1. Теоретичний матеріал

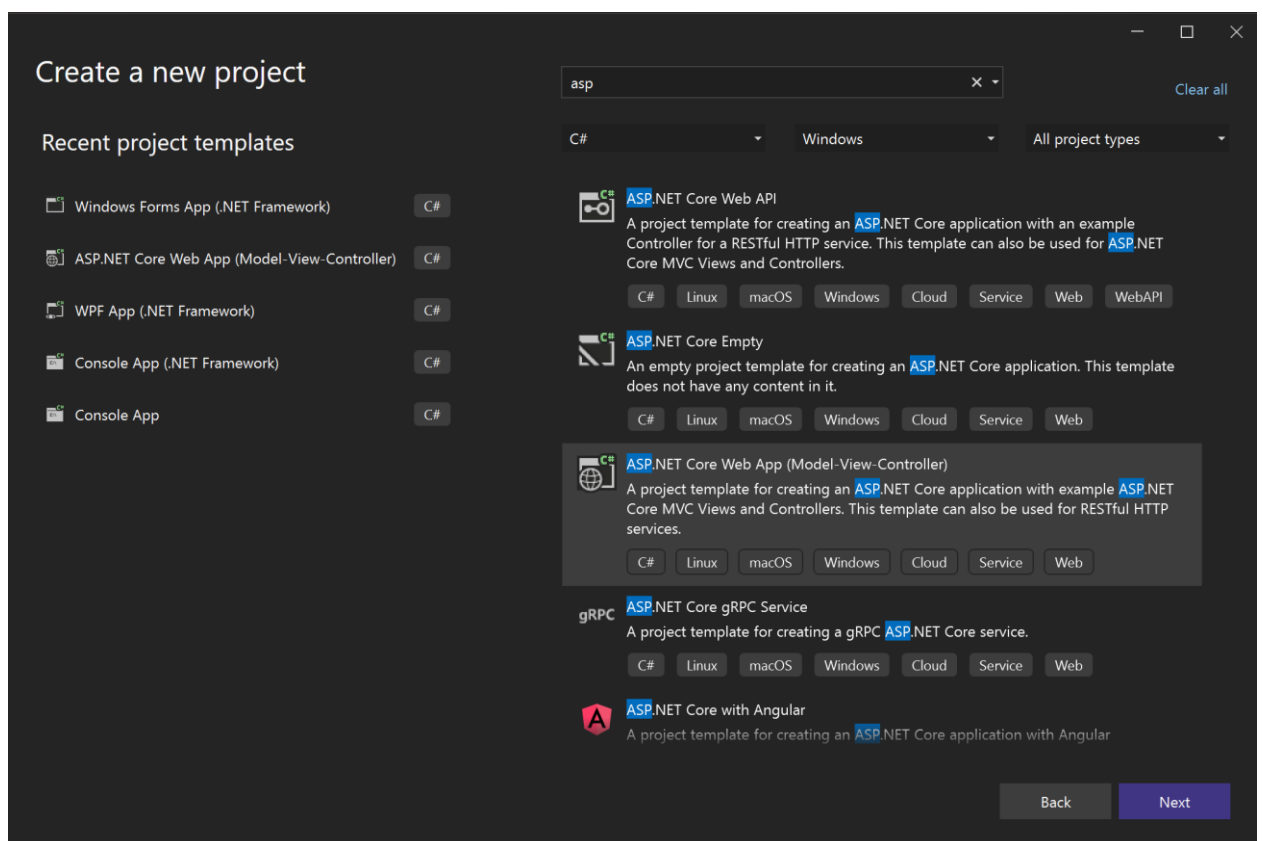
ASP.NET Core – це крос-платформний фреймворк з відкритим вихідним кодом для розробки веб-додатків, який можна використовувати для швидкого створення динамічних програм з відтворенням на стороні сервера.

ASP.NET Core має високий ступінь сумісності концепцій з ASP.NET MVC, який об'єднує функціональність MVC, Web API та Web Pages. В попередніх версіях платформи дані технології реалізовані окремо і тому містять багато дублювальної функціональності. Тепер це об'єднано в одну програмну модель ASP.NET Core MVC. Вебформи повністю вийшли в минуле.

ASP.NET Core MVC є спрощеною, ефективно тестованою платформою з відкритим вихідним кодом, оптимізованою для використання з ASP.NET Core. ASP.NET Core MVC надає на основі шаблонів спосіб створення динамічних веб-сайтів з чітким поділом завдань.

Розглянемо основні кроки для створення проекту за технологією Asp.Net Core MVC.

1. Вибір шаблону проекту, що створюється. Обираємо шаблон ASP.NET Core Web App (Model – View - Controller)



2. Вказуємо ім'я проекту та його розташування

Configure your new project

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Project name
AspNetCoreMvc

Location
D:\WorkProjects\C#\My

Solution name ⓘ
AspNetCoreMvc

☐ Place solution and project in the same directory

Project will be created in "D:\WorkProjects\C#\My\AspNetCoreMvc\AspNetCoreMvc\"

Back Next

3. На наступному кроці нічого не міняємо, тобто, натискаємо кнопку Create:

Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Framework ⓘ
.NET 6.0 (Long Term Support)

Authentication type ⓘ
None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ
Linux

☐ Do not use top-level statements ⓘ

Back Create

4. Visual Studio формує проект.

Склад проекту можна переглянути у вікні Solution Explorer:

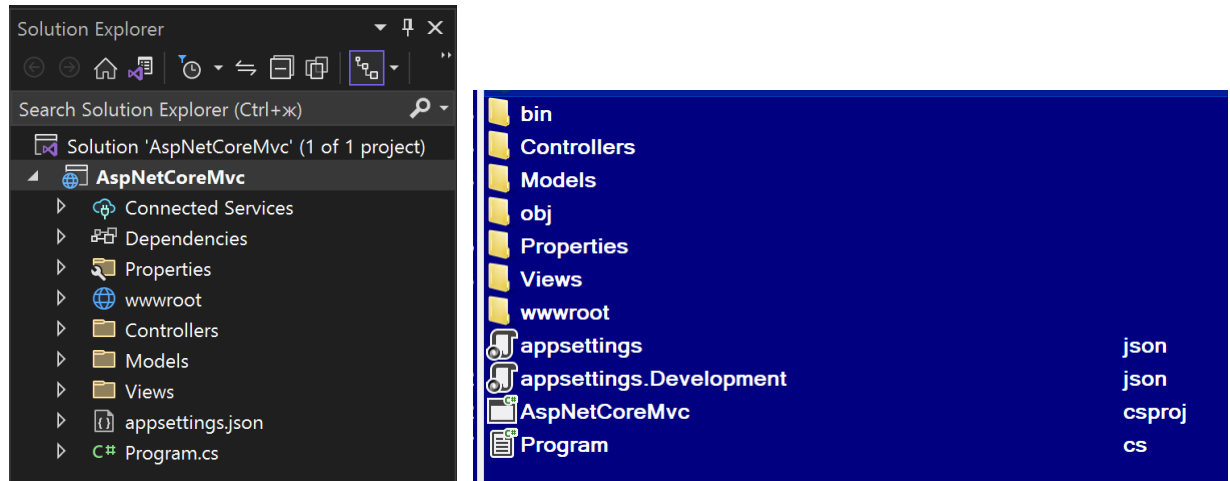


Рисунок 3.1 – Склад проекту у вікні Solution Explorer та на жорсткому диску

Модель архітектури MVC поділяє додаток на три основні компоненти: Model (модель), View (подання) та Controller (контролер).

За допомогою моделі MVC можна створювати програми, які зручніше тестувати та оновлювати порівняно з традиційними монолітними програмами.

Програми на основі моделі MVC містять таке:

- Моделі: класи, які подають дані у додатку. Класи моделі використовують логіку перевірки, що дозволяє застосовувати бізнес-правила до цих даних. Як правило, об'єкти моделі витягують та зберігають стан моделі у базі даних.
- Представлення: компоненти, які формують інтерфейс програми користувача. Як правило, в інтерфейсі користувача відображаються дані моделі.
- Контролери: класи, які:
 - обробляють запити браузера;
 - одержують дані моделі;
 - викликають шаблони представлення дзвінків, які повертають відповідь.

Відносини між компонентами патерну можна описати такою схемою:



Рисунок 3.2 – Схема відносин між компонентами патерну MVC

У програмі MVC представлення використовується лише для відображення інформації. Контролер обробляє та реагує на введення та взаємодію користувача. Наприклад, контролер обробляє сегменти URL-адреси та значення рядка запиту і передає ці значення моделі. Модель може використовувати ці значення для виконання запитів до бази даних.

Розглянемо призначення папок та файлів проекту:

/Controllers	Сюди розміщуються класи контролерів
/Models	Сюди розміщуються класи моделей уявлень та моделей предметної області
/Views	У цій папці зберігаються уявлення та часткові уявлення, зазвичай згруповані разом у папках з іменами контролерів, з якими вони пов'язані
/Views/Shared	У цій папці зберігаються компонування та уявлення, які не є специфічними для будь-якого контролера
Program.cs	Керує запуском та налаштуванням вашої програми під час виконання
Назва проекту.csproj	Містить всю необхідну інформацію для складання проекту, включаючи пакети NuGet
/Properties	Містить один файл, launchSettings.json, який керує запуском програми з Visual Studio та її налагодження.
/wwwroot	Єдина папка в програмі, до якої браузерам дозволено прямий доступ під час перегляду веб-програми. Тут програма може зберігати свої файли CSS, JavaScript, зображення або статичні HTML-файли, а браузери зможуть отримувати доступ до них. Вони не зможуть отримати доступ до файлів, що знаходяться за межами папки wwwroot
appsettings.json та appsettings.Development.json	Надають параметри конфігурації, які використовуються під час виконання для керування поведінкою програми.

Хоча папки wwwroot і Properties є на диску, видно, що оглядач рішень показує їх як спеціальні вузли, розташовані не за алфавітом, у верхній частині вашого проекту. У вас є ще два спеціальні вузли в проекті, Dependencies та Connected Services, але у них немає відповідних папок на диску. Вони показують всі залежності проекту, такі як пакети NuGet та віддалені служби.

Найважливіший файл у вашому проекті – Назва проекту.csproj, оскільки він описує, як здійснити складання проекту. Visual Studio не відображає файл .csproj явно, але його можна відредагувати, двічі клацнувши на ім'я проекту в браузері рішень. Наприклад, у цьому файлі описується версія .Net платформи, яка підключена до вашого проекту:

```

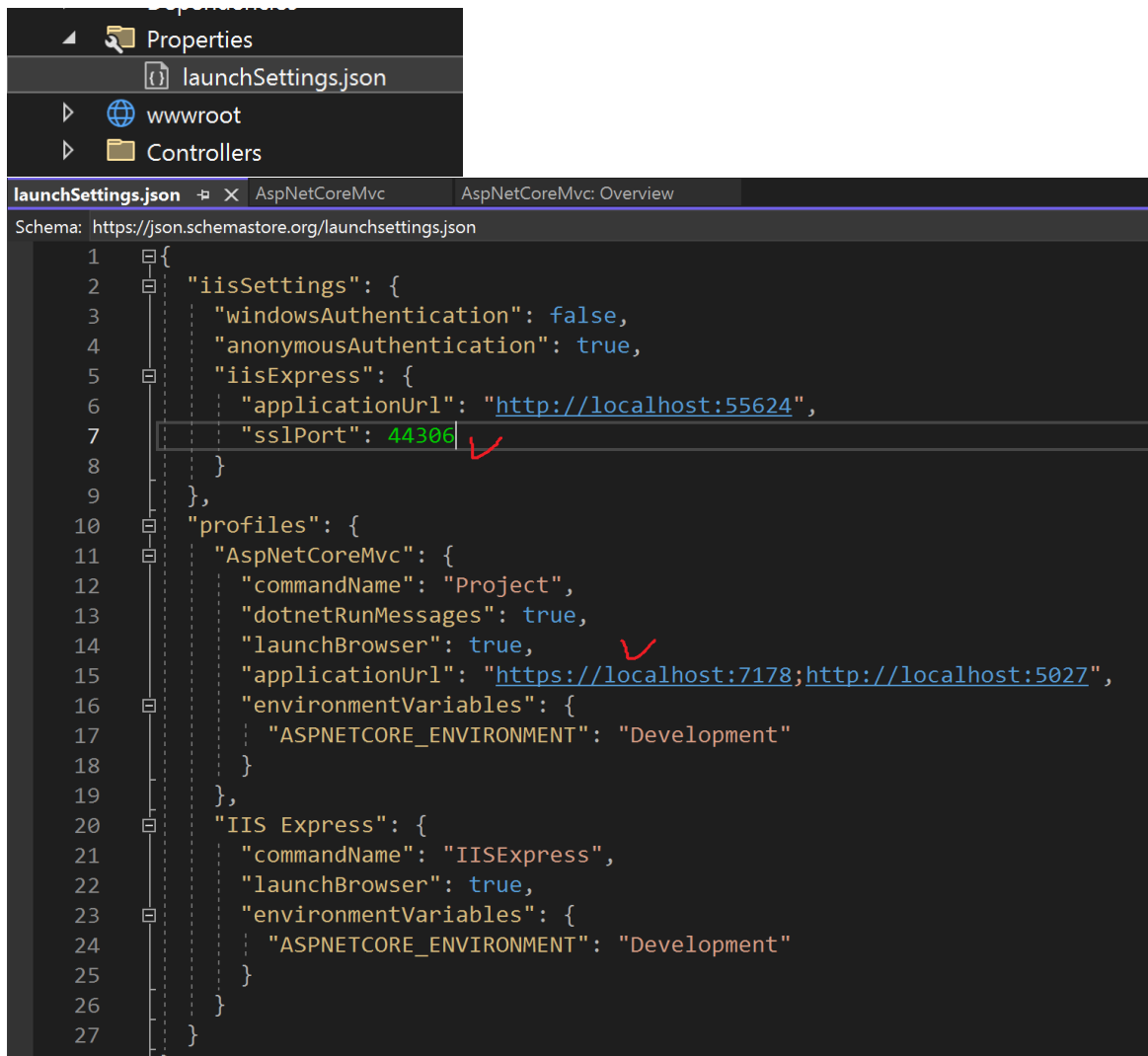
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>
</Project>

```

Файл проекту .csproj показує набір засобів розробки, цільовий фреймворк та посилання. Для простих програм вам, ймовірно, не потрібно сильно змінювати файл проекту. Атрибут Sdk в елементі Project включає параметри за замовчуванням, які описують, як зібрати проект, тоді як елемент TargetFramework описує фреймворк, на якому запуститься ваша програма.

Найчастіші зміни, які ви вносите у файл проекту, – це додавання додаткових пакетів NuGet за допомогою елемента PackageReference. За замовчуванням у програмі немає посилань на пакети NuGet.

Якщо вибрати файл launchsettings.json у групі Properties, цей файл містить налаштування, наприклад номер порту та адресу нашого вебсайту (рядом також розташовується додаткова версія адресу вебсайту):



Розглянемо зміст файлу Program.cs:

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days.
    // You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
```

```
app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

`builder.Services.AddControllersWithViews()` – додає підтримку контролерів з уявленнями

`app.UseHttpsRedirection()` – цей компонент гарантує, що ваша програма відповідає тільки на безпечні (HTTPS) запити, що є загальноприйнятою галузевою практикою.

`app.UseStaticFiles()` – цей компонент відповідає за обробку запитів до статичних файлів, таких як файли CSS, файли JavaScript та зображення. Коли запит надходить у компонент, він перевіряє, чи цей запит відноситься до існуючого файлу. Якщо це так, компонент повертає файл. Якщо ні, запит ігнорується, і наступний компонент може спробувати обробити запит.

`app.UseRouting()` – додає проміжне ПЗ маршрутизації кінцевих точок, яке визначає, яку кінцеву точку виконати.

`app.UseAuthorization()` – додає проміжне ПЗ авторизації, яке при необхідності може блокувати доступ до певних сторінок.

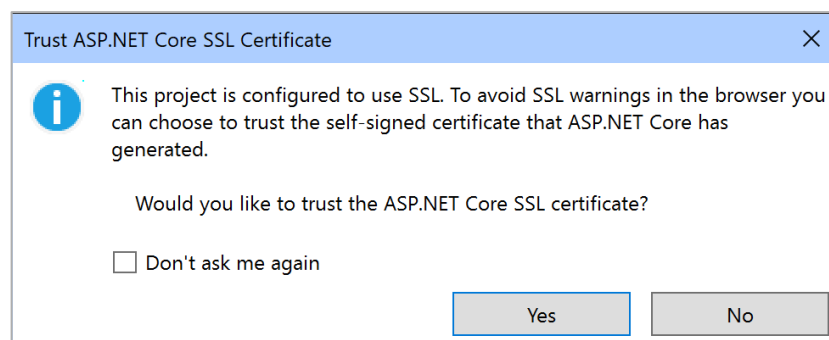
Для додавання маршруту, який буде зіставлятися з методами контролерів, у об'єкта `IEndpointRouteBuilder` (як який виступає об'єкт `WebApplication`) викликається метод `MapControllerRoute()`. Він визначає один маршрут:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

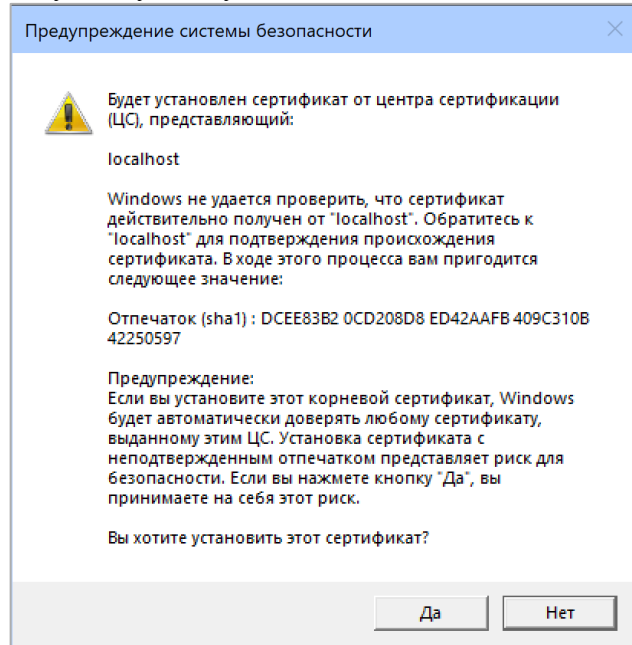
Параметр `name` визначає назву маршруту – вона довільна і у разі має значення "default". А параметр `pattern` визначає шаблон маршруту, з яким зіставлятимуться вхідні маршрути.

Шаблон маршруту використовує три параметри. Параметр "controller" буде зіставлятися на ім'я з одним із контролерів програми, а параметр "action" – з дією цього контролера. Наприклад, при запиті `http://localhost:3456/Home/Index` система вибере для обробки запиту контролер `Home` – ім'я контролера без префікса `Controller` та його дію `Index`.

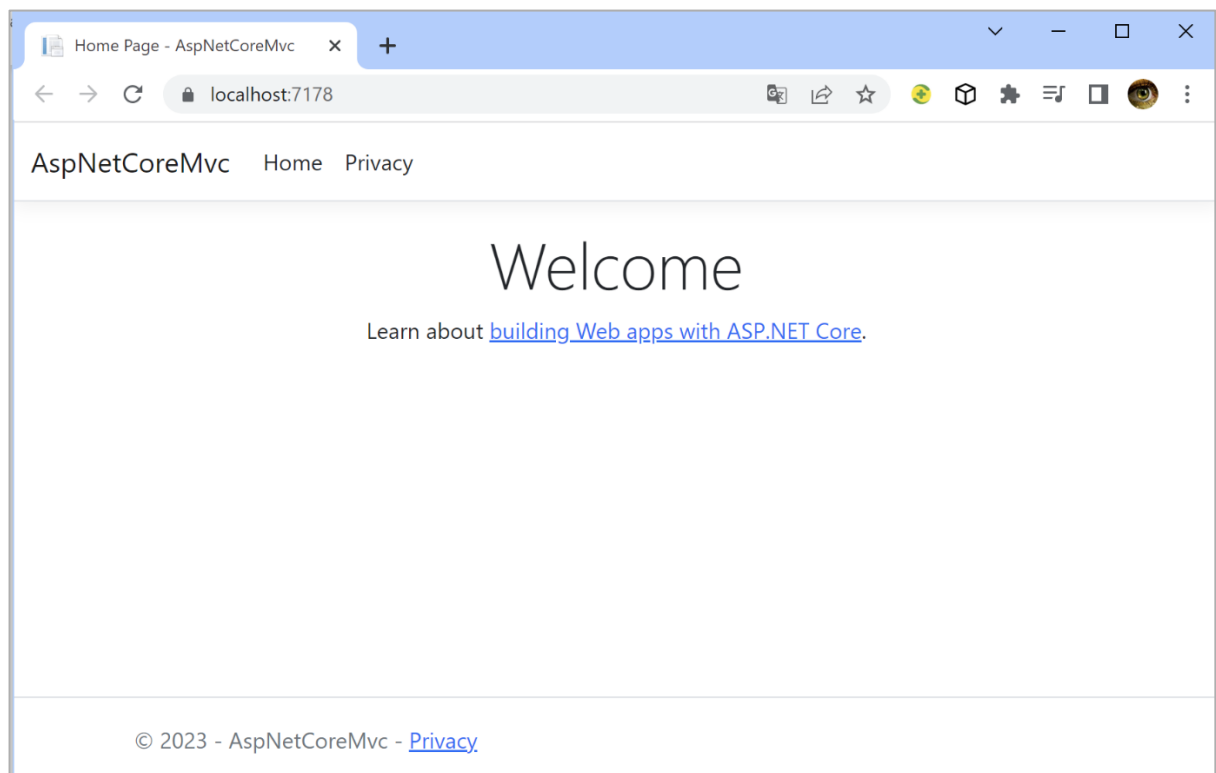
Запустимо проект. При запуску проекту з'явиться вікно для підтвердження нашої довіри до сертифіката SSL.



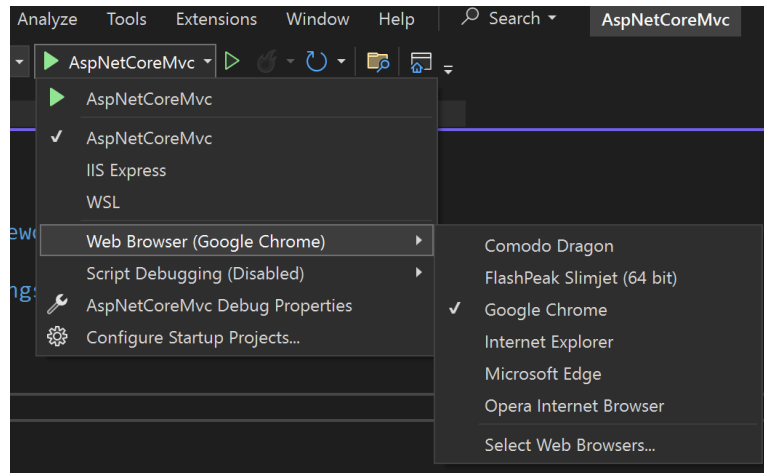
У цьому вікні ми підтверджуємо нашу довіру до сертифікату, натискаючи кнопку Yes. Також натискаємо «Так» у наступному вікні:



В результаті запуску проекту створиться веб-сторінка:

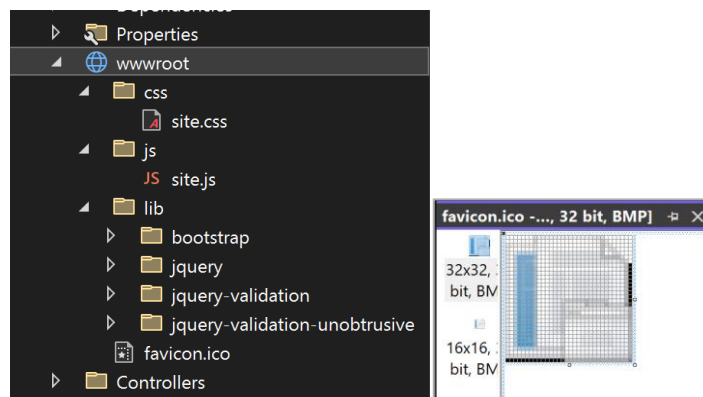


Перед запуском проекту можна уточнити в якому браузері бажаємо відкрити сторінку:

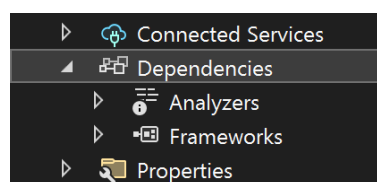


Розглянемо структуру створеного додатку.

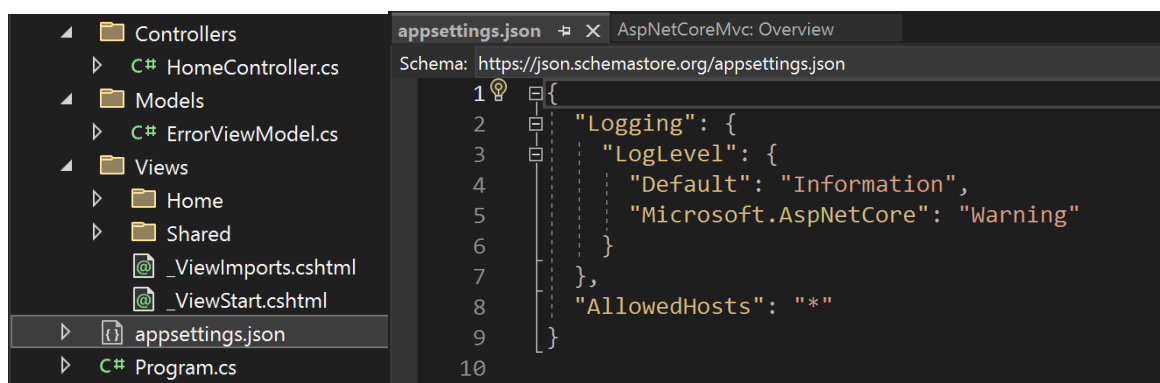
Wwwroot міститиме файли користувача: стилі, java-script файли, картинки, відео, звукові ефекти, документи тощо. Файл favicon.ico – це зображення, яке відображається поряд із заголовком сторінки.



У папці Dependencies (Залежності) будуть встановлені всі бібліотеки внутрішнього пакетного менеджера NuGet.



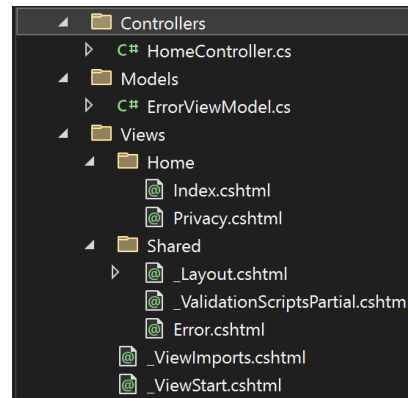
У файлі appsettings.json містяться установки, наприклад, можна вказати підключення до бази даних.



З файлу Program.cs відбувається запуск усього проекту. В коді файлу Program.cs вказується який контролер та сторінка запускаються за замовчанням:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Розглянемо що зберігається в папках Model, View та Controller.



Компонента Модель (model) патерна MVC описує дані, що використовуються в додатку, а також логіку, яка пов'язана безпосередньо з даними, наприклад, логіку валідації даних. Як правило, об'єкти моделей зберігаються у базі даних. Склад цієї компоненти зберігається в папці Model.

Контролери – це компоненти для керування взаємодією з користувачем, роботи з моделлю та вибору подання для відображення. У MVC представлення служить тільки для відображення інформації. Обробку введених даних, формування відповіді та взаємодію з користувачем забезпечує контролер. Склад цієї компоненти зберігається в папці Controller. В структурі проекту можна побачити створений контролер HomeController та пов'язану з ним папку View\Home.

Контролер HomeController представлений папкою Home всередині папки Views. Папка Home містить уявлення веб-сторінок Index (домашня сторінка), Privacy. Коли користувач запитує одну з цих двох веб-сторінок, дії контролера в Home контролері визначають, яке з двох уявлень використовується для створення та повернення веб-сторінки користувачеві.

```
public IActionResult Index()
{
    return View();
}

public IActionResult Privacy()
{
    return View();
}
```

Рисунок 3.3 – Код в файлі HomeController.cs

В ASP.NET MVC Core уявлення – це файли з розширенням cshtml, які містять код інтерфейсу в основному на мові html, а також конструкції Razor – спеціального движка уявлень, який дозволяє переходити від коду html до коду мовою C#.

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.
</div>
```

```
@{
    ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>
```

Рисунок 3.4 – Код в файлах Index.cshtml та Privacy.cshtml

Папка Shared зберігає загальні уявлення всіх контролерів. За замовчуванням це файли _Layout.cshtml (використовується як майстер-сторінки), Error.cshtml (використовуються для відображення помилок) та _ValidationScriptsPartial.cshtml (часткове уявлення, яке включає скрипти валідації форми).

Файли Index.cshtml та Privacy.cshtml вбудовуються в файл _Layout.cshtml, що в результаті дає той вигляд сторінки, що ми бачим в браузері. Тобто в файлі _Layout.cshtml описуємо код, який повинен з'являтися во всіх сторінках нашого проекту. Місце куди буде вбудовуватися код зі сторінок, наприклад Index.cshtml та Privacy.cshtml помічено в файлі _Layout.cshtml методом @RenderBody(), який є плейсхолдером.

```
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
```

Рисунок 3.5 – Фрагмент коду з файлу _Layout.cshtml

Крім методу RenderBody(), який вставляє основний вміст уявлень, майстер-сторінок може використовувати спеціальний метод RenderSection() для вставки секцій. Майстер-сторінка може мати декілька секцій, куди уявлення можуть помістити свій вміст.

В файлі _Layout.cshtml є ще одне місце для вбудовування інформації для відображення заголовку сторінки. Це конструкція ViewData куди передається назва сторінки з відповідного файлу:

```
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - ASP.NET Core MVC</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/AspNetCoreMvc.styles.css" asp-append-version="true" />
</head>
```

ViewData представляє словник з пар ключ-значення:

У корені каталогу Views також можна знайти два файли _ViewImports.cshtml та _ViewStart.cshtml. Ці файли містять код, який автоматично додається до всіх уявлень. _ViewImports.cshtml встановлює деякі спільні для всіх уявлень простору імен, а _ViewStart.cshtml встановлює спільну майстер-сторінку. Тобто файл _ViewStart.cshtml показує який файл використовується для вбудовування інформації та коду сторінок:

```
@{
    Layout = "_Layout";
}
```

Рисунок 3.6 – Код з файлу _ViewStart.cshtml

Змінимо зовнішній вигляд наших сторінок.

Видалимо сторінку Privacy, для цього видалимо файл Privacy.cshtml та в файлі HomeController.cs видалимо метод Privacy().

Відредагуємо файл _Layout.cshtml. По-перше, відредагуємо заголовок: рядок з назвою сторінок та видалимо перший та третій рядок з тегом link. Тоді заголовок повинен мати наступний вигляд:

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"]</title>
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
```

Далі видалимо все, що знаходиться всередині тегу body, та створимо всередині таку конструкцію:

```
<header>                                Верхній колонтитул для всіх сторінок
</header>

<main>                                  Тут буде інформація зі сторінок
</main>

<footer>                                Нижній колонтитул для всіх сторінок
</footer>
```

Наприклад, будемо створювати веб-додаток для кафедри. Тоді всередині тегу body помістимо таку інформацію:

```
<body>
    <header>
        <span class="logo">Кафедра кібербезпеки та захисту інформації</span>
        <nav class="menu">
            <a href="/">-Загальні відомості-</a>
            <a href="/contacts">-Зворотній зв'язок-</a>
        </nav>
    </header>

    <main>
        @RenderBody()
    </main>

    <footer>&copy; 2026 </footer>
</body>
```

Далі відредагуємо файл Index.cshtml, видаливши всі рядки крім перших трьох, де формується назва сторінки. Змінимо назву сторінки. Додамо деяку інформацію для цієї сторінки. В результаті файл Index.cshtml буде мати наступний вигляд:

```
@{
    ViewData["Title"] = "Кафедра КБЗІ";
}

<div>
    <h3>Загальна інформація</h3>
    Кафедра кібербезпеки та захисту інформації була заснована у 2025 році у складі
    Навчально-наукового інституту інформаційних технологій та інноваційного підприємництва,
    яка стала випусковою для бакалаврів, магістрів в галузі
    інформаційних технологій. Очолила кафедру <b>д.т.н., проф. Кобозєва А.А.</b>
    <br /><br>
    Кафедра КБЗІ здійснює підготовку фахівців за такими спеціальностями та освітніми програмами:
    <br />
    <i>
        Спеціальність F5 «Кібербезпека та захист інформації» (освітня програма «Кібербезпека та захист інфор
    </i>
    <br /><br>
    Освітня програма «Кібербезпека та захист інформації» спрямована на підготовку фахівців з
    інформаційної безпеки, зокрема в сфері критичної інфраструктури держави.
    Характерною особливістю програми є надання можливості поглибленого вивчення дисциплін,
    присвячених розробці та використанню спеціалізованого програмного забезпечення,
    зокрема
    <font color="Red">
        для виявлення фейкових цифрових контентів (зображень, відео, аудіо),
        для організації та виявлення прихованого каналу зв'язку в каналі загального користування тощо.
    </font>
</div>
```

Якщо запустити наш додаток, то сайт буде мати наступний вигляд:

Кафедра кібербезпеки та захисту інформації
[-Загальні відомості-](#) [-Зворотній зв'язок-](#)

Загальна інформація

Кафедра кібербезпеки та захисту інформації була заснована у 2025 році у складі Навчально-наукового інституту інформаційних технологій та інноваційного підприємництва, яка стала випусковою для бакалаврів, магістрів в галузі інформаційних технологій. Очолила кафедру **д.т.н., проф. Кобозєва А.А.**

Кафедра КБЗІ здійснює підготовку фахівців за такими спеціальностями та освітніми програмами:
Спеціальність F5 «Кібербезпека та захист інформації» (освітня програма «Кібербезпека та захист інформації»)

Освітня програма «Кібербезпека та захист інформації» спрямована на підготовку фахівців з інформаційної безпеки, зокрема в сфері критичної інфраструктури держави. Характерною особливістю програми є надання можливості поглибленого вивчення дисциплін, присвячених розробці та використанню спеціалізованого програмного забезпечення, зокрема **для виявлення фейкових цифрових контентів (зображень, відео, аудіо), для організації та виявлення прихованого каналу зв'язку в каналі загального користування тощо.**

© 2026

Для того щоб прикрасити зовнішній вигляд сторінки знадобиться відредагувати файл site.css (wwwroot\css).

```
header .logo{
    font-weight: 600;
    font-size: 25px;
}

header nav a {
    color: #1002ef;
    text-decoration: none;
    margin-left: 15px;
}

main {
    width: 80%;
    padding: 10px 10%;
    margin: 30px 0;
}

footer {
    margin-top: 150px;
    text-align: center;
    width: 100%;
    padding: 15px 0;
    color: #54998c;
    background: #b6dbbe;
    border-top: 2px solid #54998c;
}
```

Рисунок 3.7 – Код з файлу site.css

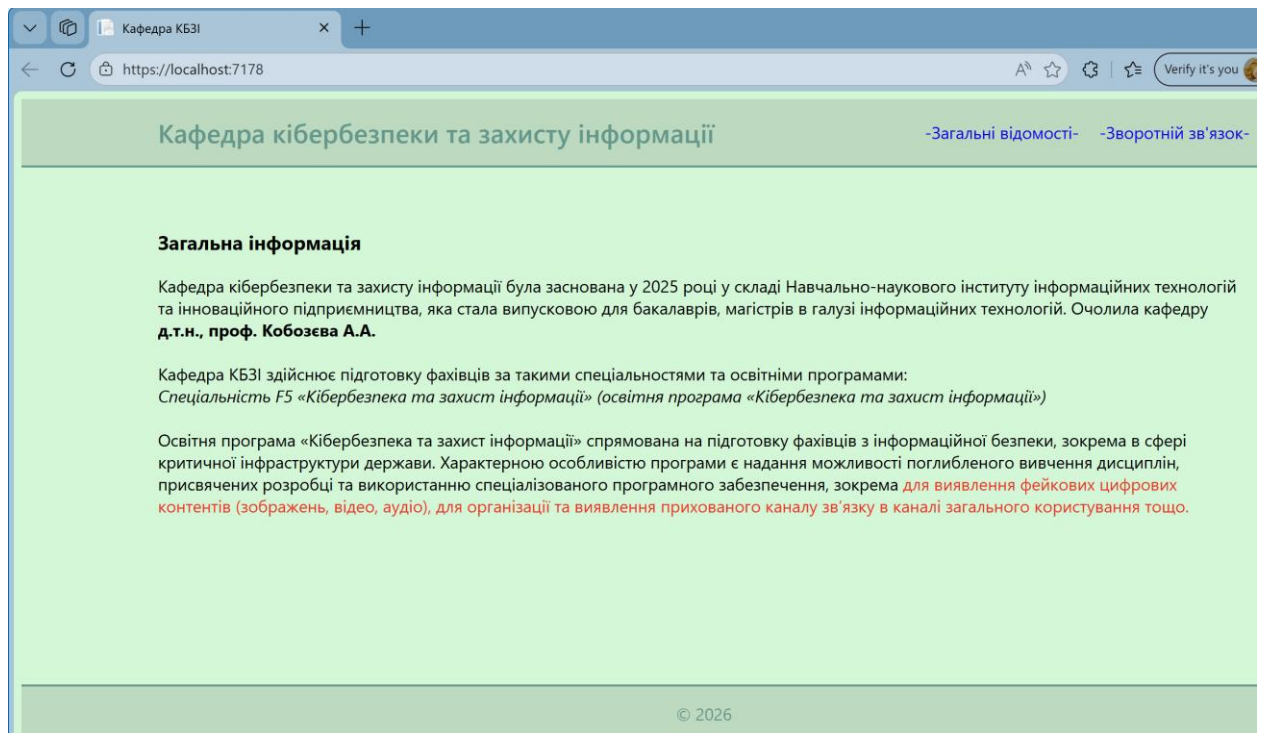


Рисунок 3.8 – Результат впровадження коду з файлу site.css

2. Завдання до практичної роботи 3

Створіть новий веб-додаток за технологією Asp.Net Core MVC для обраної тематики. Розробіть та створіть вміст файлів Index.cshtml та site.css за обраною тематикою.

Практична робота 4. Створення нових сторінок у веб-проекті

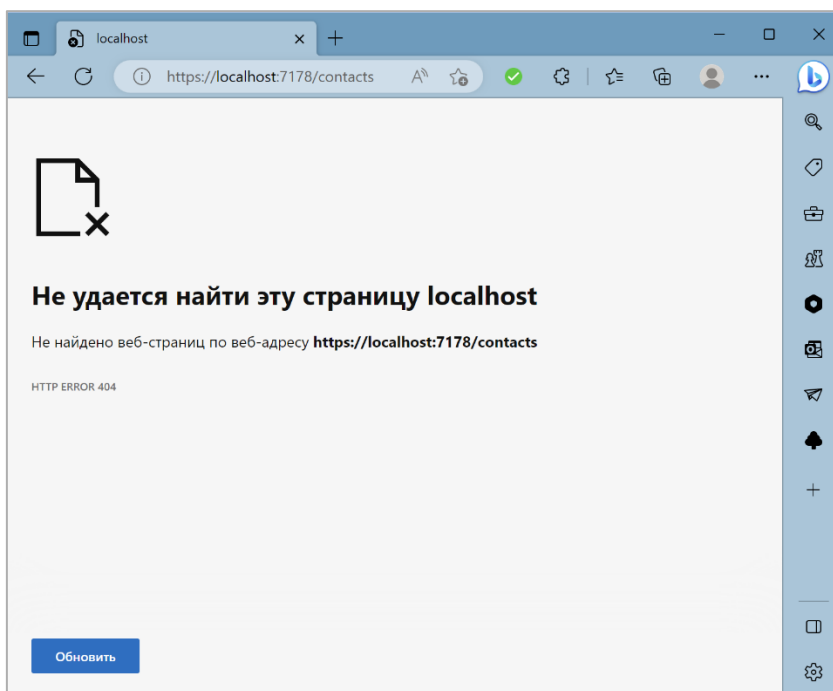
Мета роботи: ознайомитися зі створенням нових контролерів. Набути практичних навичок в створенні нових контролерів та відповідних файлів представлення.

1. Теоретичний матеріал

На попередньому лабораторному занятті ми створили головну сторінку нашого веб-додатку та записали у файлі `_Layout.cshtml` перехід на іншу сторінку «-Зворотній зв'язок-».

```
<a href="/contacts">-Зворотній зв'язок-</a>
```

Якщо запустити наш проект та перейти за посиланням на сторінку «-Зворотній зв'язок-», то побачимо наступне:



Якщо подивитися на url, то він має наступний вигляд: «`https://localhost:7178/contacts`».

Маршрутизація полягає в тому, щоб виявити ім'я класу контролера, який необхідно створити, та метод дії, який потрібно виконати для генерації HTTP-відповіді.

Маршрут за замовчуванням налаштований для MVC, як показано нижче:

```
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Частини маршруту, записані у фігурних дужках {}, називаються сегментами і подібні до іменованих параметрів методу. Значення цих сегментів може бути будь-яким рядком.

Цей шаблон передбачає аналіз усіх URL, введених користувачем в адресному рядку, та їх розбір для вилучення імен контролера та дії, а також опціонального значення id (таке воно завдяки символу ?).

Якщо користувач не вказав ці дані, то застосовуються значення за замовчуванням: `Home` як контролер і `Index` як дія (присвоєння через `=` встановлює значення за замовчуванням для сегмента з ім'ям).

У таблиці 2.1 наведено приклади URL і те, як маршрут за замовчанням визначає імена контролера та дії.

Таблиця 2.1

URL-адреса	Контролер	Дія	ID
/	Home	Index	—
/Muppet	Muppet	Index	—
/Muppet/Kermit	Muppet	Kermit	—
/Muppet/Kermit/Green	Muppet	Kermit	Green
/Products	Products	Index	—
/Products/Detail	Products	Detail	—
/Products/Detail/3	Products	Detail	3

Сегменти в URL не чутливі до регістру.

Основним елементом архітектури ASP.NET Core MVC є контролер. При отриманні запиту система маршрутизації вибирає обробки запиту потрібний контролер і передає йому дані запиту. Контролер обробляє ці дані та посилає назад результат обробки.

За маршрутом та вхідним URL ASP.NET Core MVC визначить ім'я контролера, тому знадобиться клас, успадкований від класу Controller.

Обов'язки контролера полягають у наступному:

- визначення у конструкторі (або конструкторах) сервісів, необхідних контролеру для того, щоб він перебував у допустимому стані та правильно функціонував;
- вилучення параметрів із HTTP-запиту;
- використання параметрів для отримання будь-яких додаткових даних, необхідних для побудови моделі подання, та передачі їх у відповідне подання для клієнта.
- повернення результатів з подання клієнту як HTTP-відповідь з відповідним кодом стану.

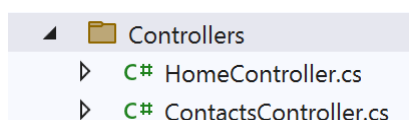
Проект, що створюється за шаблоном ASP.NET Core Web App (Model-View-Controller), за замовчуванням містить щонайменше один контролер – клас HomeController.

При використанні контролерів є деякі умовності. Насамперед зазвичай у проекті контролери розміщуються в каталог Controllers. Однак це в принципі необов'язково - можна додавати контролери до інших папок або навіть у корінь проекту.

Для роботи зі сторінкою contacts нам не достатньо мати контролер HomeController. Якщо ми бажаємо, щоб контролер HomeController обробляв сторінку contacts, то url повинен мати інший вигляд, наприклад для нашого прикладу це буде «https://localhost:7178/home/contacts». Якщо ми бажаємо залишити url таким як він зараз є, то необхідно створити новий контролер.

В ASP.NET Core MVC контролер представляє звичайний клас мовою C#, який зазвичай успадковується від абстрактного базового класу Microsoft.AspNetCore.Mvc.Controller і який, як і будь-який клас мовою C#, може мати поля, властивості, методи. Згідно з угодами про назву назви контролерів зазвичай закінчуються на суфікс "Controller", решта ж до цього суфікса вважається ім'ям контролера.

Додаємо в папку Controllers новий контролер. Для цього виділяємо назву папки та натискаємо праву кнопку миші, обираємо Add → Controller. З'явиться вікно для вибору контролера (рисунк 2.1). На наступному вікні пропонується дати назву контролеру, що створюється. Введемо назву «ContactsController.cs», причому слово Controller категорично не можна видаляти з назви контролера, так як ASP.NET тоді буде сприймати файл як простий клас. Натиснути Додати. В результаті у проект додається новий контролер:



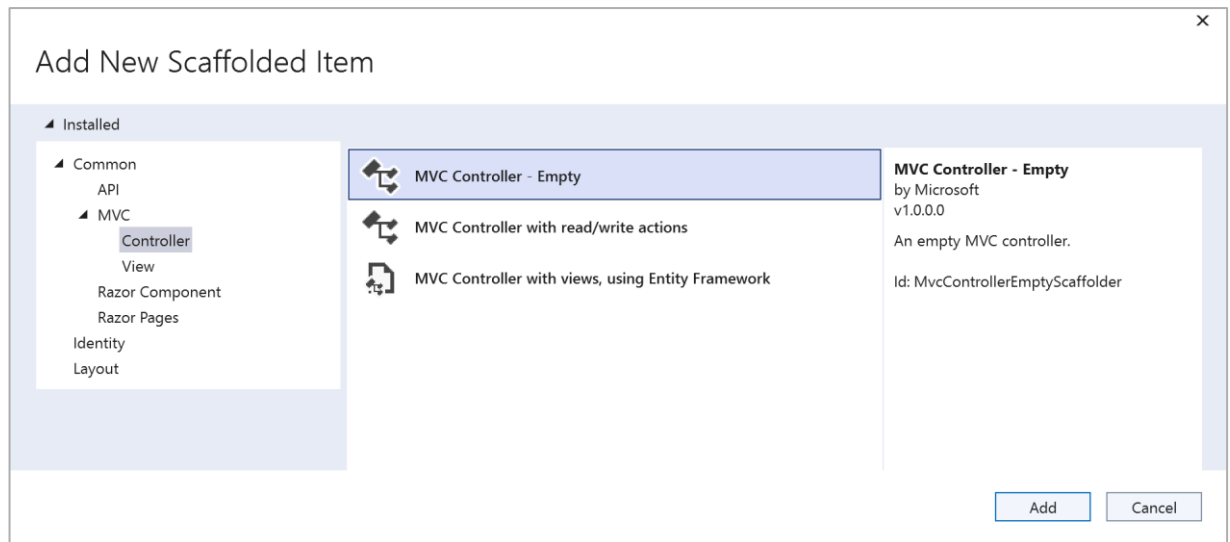


Рисунок 4.1 – Вікно додавання нового контролера

Файл `ContactsController.cs` має наступний вигляд:

```
using Microsoft.AspNetCore.Mvc;

namespace AspNetCoreMvc.Controllers
{
    public class ContactsController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Ключовим елементом контролера є його дії. Дії контролера – це `public` методи, які можуть зіставлятися із запитам.

Створений клас має зараз один метод `Index()`, який буде запускатися кожного разу як ми будемо переходити на сторінку `contacts`. При переході на сторінку `contacts` буде визиватися шаблон з папки `contacts` та шаблон повинен мати ім'я `Index`. Зараз у нас не має такої папки та шаблону, тож нам потрібно їх створити.

Перед цим розглянемо декілька властивостей та можливостей що до використання контролерів.

При зверненні до контролера середовище `ASP.NET` створює для цього контролера контекст, який містить різні пов'язані з контролером дані. Для отримання контексту в класі контролера нам доступна властивість `ControllerContext`, яка представляє однойменний клас `ControllerContext`. Цей клас визначає ряд важливих властивостей:

- `HttpContext`: містить інформацію про контекст запиту;
- `ActionDescriptor`: повертає дескриптор дії – об'єкт `ActionDescriptor`, який описує контролера, що викликається;
- `ModelState`: повертає словник `ModelStateDictionary`, який використовується для валідації даних, надісланих користувачем;
- `RouteData`: повертає дані маршруту.

Об'єкт `HttpContext` інкапсулює всю інформацію про запит. Зокрема, він визначає такі властивості:

- `Request`: містить інформацію про поточний запит;
- `Response`: управляє відповіддю;

- User: представляє поточного користувача, який звертається до програми;
- Session: об'єкт для роботи із сесіями.

Розглянемо деякі з цих властивостей.

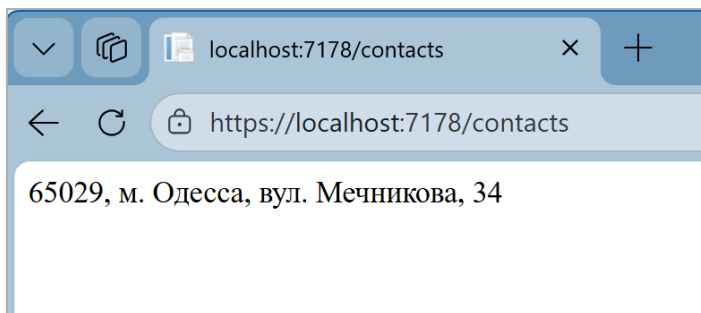
Властивість `HttpContext.Response` представляє об'єкт `HttpResponse` і дозволяє керувати відповіддю на запит, зокрема, встановлювати заголовки відповіді, куки, надсилати у вихідний потік певну відповідь. Цей об'єкт доступний через властивість `Response` класу `Controller`. Серед властивостей об'єкта `Response` можна виділити такі:

- `Body`: об'єкт `Stream`, який застосовується для надсилання даних у відповідь користувачу;
- `Cookies`: куки, що надсилаються у відповіді;
- `ContentType`: MIME-тип відповіді;
- `Headers`: колекція заголовків відповіді;
- `StatusCode`: статусний код відповіді.

За допомогою об'єкта `Response` ми можемо надіслати параметри відповіді та відправити його клієнту.

Наприклад, відправимо з дії контролера дані клієнту:

```
public class ContactsController : Controller
{
    public async Task Index()
    {
        Response.ContentType = "text/html;charset=utf-8";
        await Response.WriteAsync("<div> 65029, м. Одесса, вул. Мечникова, 34 </div>");
    }
}
```



У цьому прикладі в методі `Index` спочатку встановлюється заголовок `Content-Type`, а потім за допомогою методу `WriteAsync()` відправляється деякий найпростіший вміст у вигляді рядка з кодом HTML. Оскільки метод `WriteAsync()` - асинхронний, то до нього можна застосувати оператор `await`, а метод дії `Index` визначений як асинхронний.

Властивість `HttpContext.Request` представляє об'єкт `HttpRequest` та надає різноманітну інформацію про запит. Цей об'єкт доступний через властивість `Request` класу контролера. Серед властивостей об'єкта `Request` можна виділити такі:

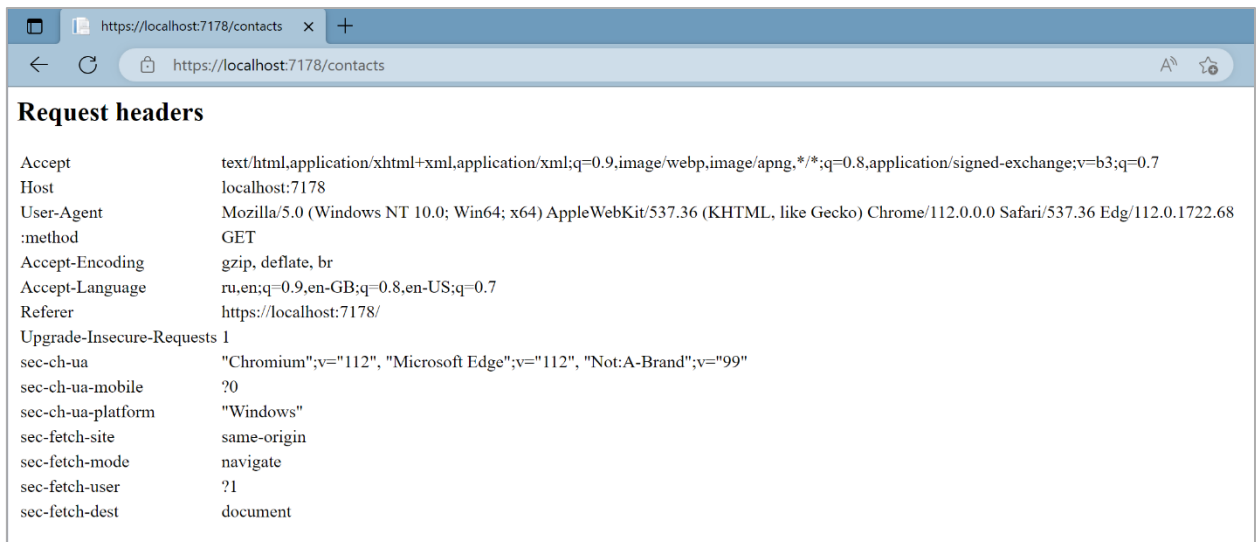
- `Body`: об'єкт `Stream`, який використовується для читання даних запиту;
- `Cookies`: куки, отримані у запиті;
- `Form`: колекція значень відправлених форм;
- `Headers`: колекція заголовків запиту;
- `Path`: повертає запитаний шлях – рядок запиту без домену та порту;
- `Query`: повертає колекцію переданих через рядок запиту параметрів;
- `QueryString`: повертає частину запиту, яка містить параметри.

Наприклад, отримаємо із запиту всі заголовки і виведемо їх у браузері:

```

public async Task Index()
{
    Response.ContentType = "text/html;charset=utf-8";
    System.Text.StringBuilder tableBuilder = new("<h2>Request headers</h2><table>");
    foreach (var header in Request.Headers)
    {
        tableBuilder.Append($"<tr><td>{header.Key}</td><td>{header.Value}</td></tr>");
    }
    tableBuilder.Append("</table>");
    await Response.WriteAsync(tableBuilder.ToString());
}

```



The screenshot shows a web browser window with the address bar displaying 'https://localhost:7178/contacts'. The page content is a table titled 'Request headers' listing various HTTP headers and their values.

Request headers	
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Host	localhost:7178
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36 Edg/112.0.1722.68
:method	GET
Accept-Encoding	gzip, deflate, br
Accept-Language	ru,en;q=0.9,en-GB;q=0.8,en-US;q=0.7
Referer	https://localhost:7178/
Upgrade-Insecure-Requests	1
sec-ch-ua	"Chromium";v="112", "Microsoft Edge";v="112", "Not:A-Brand";v="99"
sec-ch-ua-mobile	?0
sec-ch-ua-platform	"Windows"
sec-fetch-site	same-origin
sec-fetch-mode	navigate
sec-fetch-user	?1
sec-fetch-dest	document

Разом із запитом додатку можуть надходити різні дані. І, щоб отримати ці дані, ми можемо використовувати різні способи. Найпоширенішим способом вважається застосування параметрів.

Передавати значення параметрів можна різними способами. При надсиланні GET-запиту значення можна передати через рядок запиту. Рядок запиту є частиною адреси, яка йде після знаку питання ? і представляє набір параметрів, де кожен параметр відокремлений від іншого за допомогою амперсанду:

назва_ресурсу?параметр1=значення1&параметр2=значення2

Наприклад, додаємо до адресу в гіперпосиланні в файлі «_Layout.cshtml» наступне:

```

<nav class="menu">
    <a href="/">-Загальні відомості-</a>
    <a href="/contacts?name=User&age=100">-Зворотній зв'язок-</a>
</nav>

```

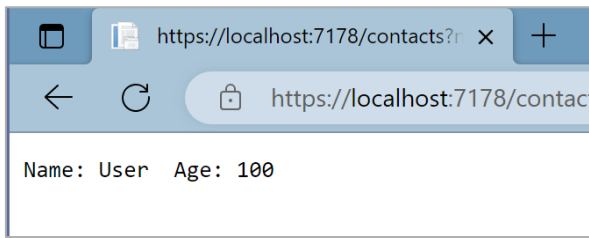
Частина «?name=User&age=100» представляє рядок запиту, який містить два параметри: name і age. Значення параметра name – "User", а значення параметра age – 100.

Наприклад, передамо метод Index через рядок запиту дані для параметра name і age:

```

public string Index(string name, int age)
{
    return $"Name: {name} Age: {age}";
}

```



Система прив'язки MVC за замовчанням порівнює параметри запиту та параметри методу на ім'я. Тобто, якщо в рядку запиту йде параметр `name`, його значення буде передаватися саме параметру методу, який також називається `name`. При цьому має бути також відповідність за типом, тобто якщо параметр методу набуває числового значення, то через рядок запиту треба передавати для цього параметра число, а не рядок.

Якщо ми не використовуємо параметри у рядку запиту, то для параметрів будуть передаватися значення за замовчанням. Тобто, якщо ми повернемо адресу в гіперпосиланні в файлі «_Layout.cshtml» у попередній вигляд:

```
<a href="/contacts">-Зворотній зв'язок-</a>
```

Та параметри «`name=User`» та «`age=100`» є параметрами за замовчанням, то маємо наступний метод `Index`:

```
public string Index(string name = "User", int age = 100)
{
    return $"Name: {name} Age: {age}";
}
```

Хоча рядок запиту переважно використовується для передачі даних примітивних типів, але ми також можемо приймати складніші об'єкти, такі як екземпляри класів, масиви, словники.

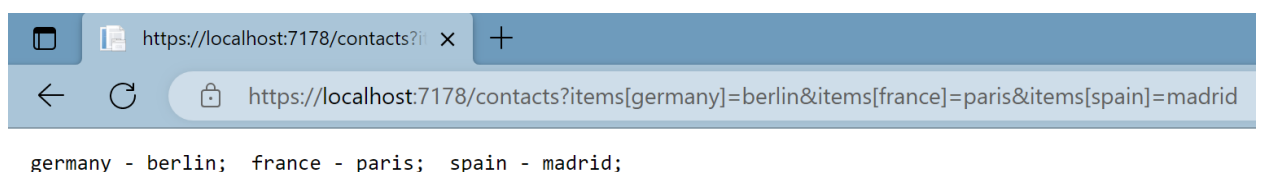
Розглянемо приклад зі словником.

```
public string Index(Dictionary<string, string> items)
{
    string result = "";
    foreach (var item in items)
    {
        result = $"{result} {item.Key} - {item.Value}; ";
    }
    return result;
}
```

Для надсилання даних ми можемо використовувати рядок запиту:

```
<a href="/contacts?items[germany]=berlin&items[france]=paris&items[spain]=madrid">-Зворотній зв'язок-</a>
```

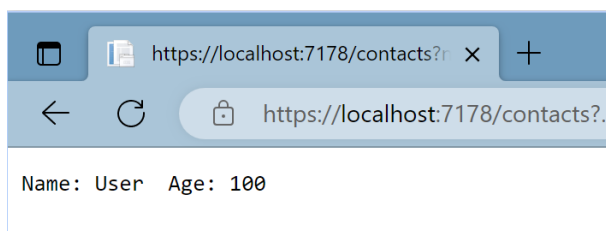
Тоді в результаті будемо мати:



Кожен окремий елемент цього рядка типу `items[germany]=berlin` представлятиме елемент словника, де `items` – назва словника, `"germany"` – ключ, а `"berlin"` – значення.

Параметри є найпростішим способом отримання даних, але насправді нам необов'язково їх використовувати. У контролері доступний об'єкт `Request`, у якого можна отримати дані рядки запиту через властивість `Request.Query`. Ця властивість представляє об'єкт `IQueryCollection`, де за ключом – назвою параметра можна отримати його значення. Наприклад:

```
public string Index()
{
    string name = Request.Query["name"];
    string age = Request.Query["age"];
    return $"Name: {name} Age: {age}";
}
```

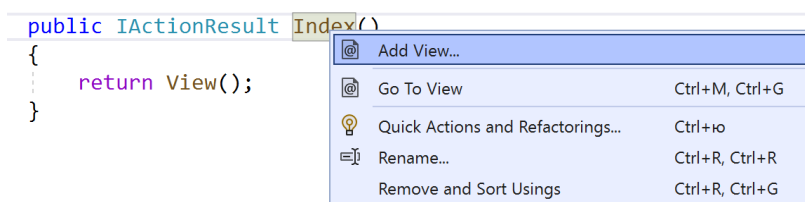


В даному випадку ми можемо передати методу `Index` дані через запит типу `https://localhost:7178/contacts?name=User&age=100`.

Повернемося до початкового варіанту методу `Index()`, який повертає `IActionResult`. Створимо для цього методу відповідне представлення.

Є декілька способів як ми можемо створити представлення створивши папки `contacts` та шаблон `Index`. Перший – в вікні `Solution Explorer` виділити папку `View` та додати до неї нову папку, а в цю папку додати новий шаблон. Розглянемо детальніше другий спосіб.

В файлі `ContactsController.cs` натискаємо правою кнопкою миші на назві методу `Index()` та обираємо `Add View`



В результаті відкриється вікно для вибору шаблону представлення (рисунок 2.2). В цьому вікні обираємо `Razor View – Empty` натискаємо `Add`. На наступному кроці (рисунок 2.3) необхідно ввести назву представлення, в нашому випадку ми залишаємо назву `«Index.cshtml»` та натискаємо кнопку `Add`.

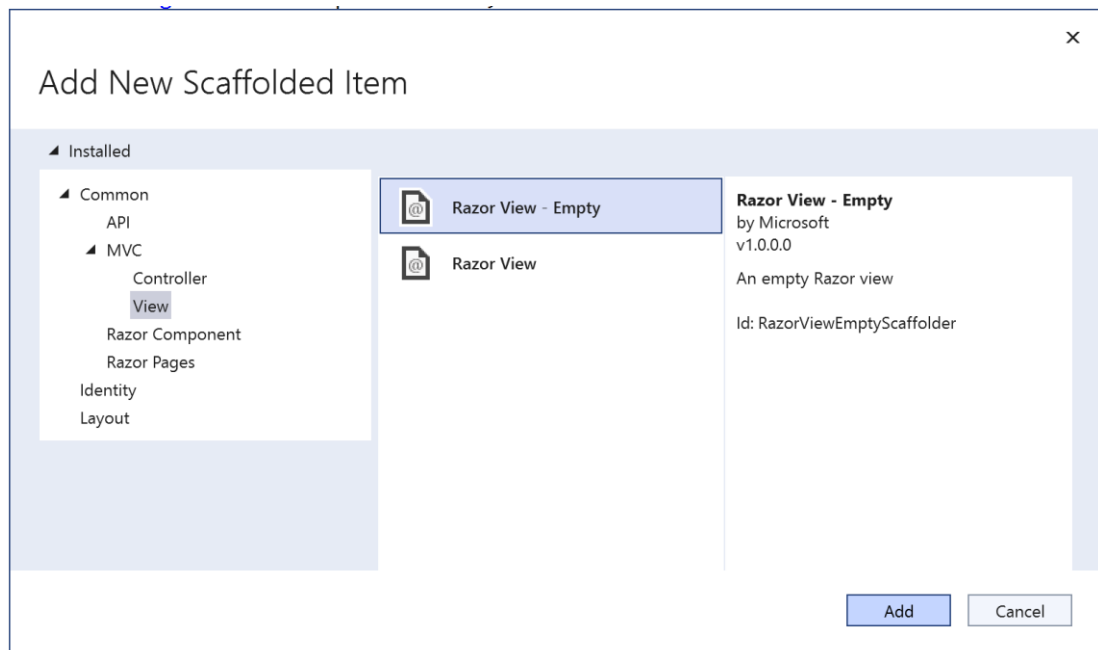


Рисунок 4.2 – Вікно для вибору шаблону представлення

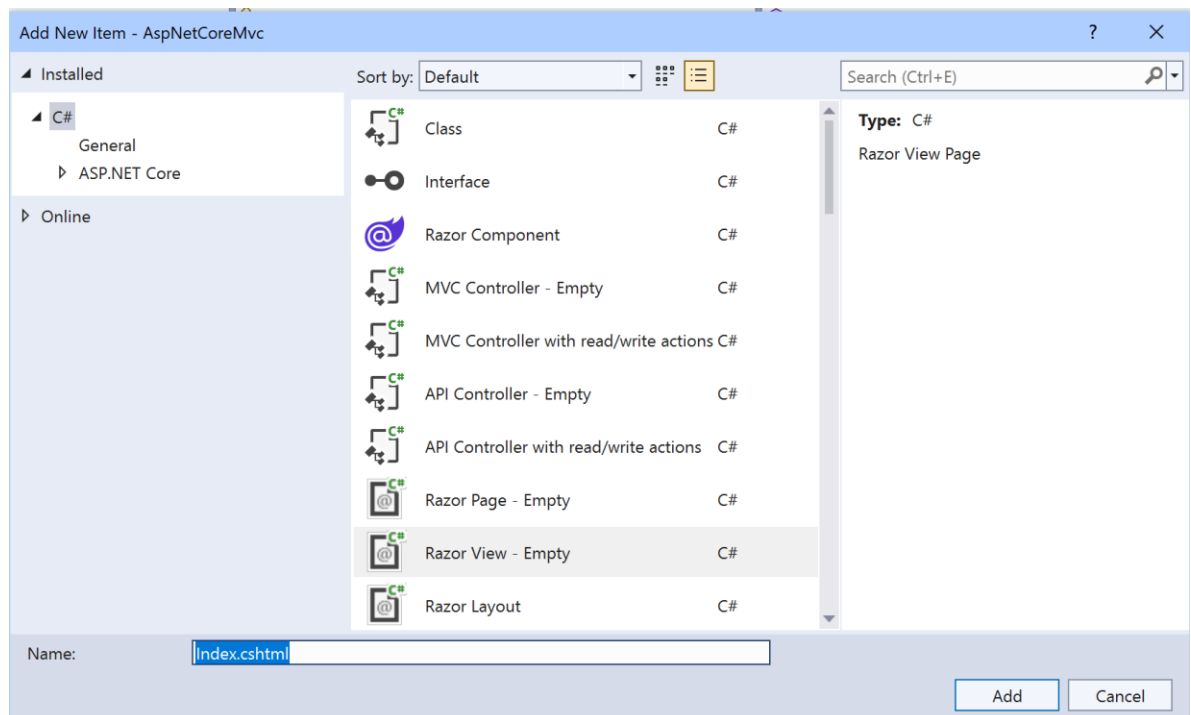
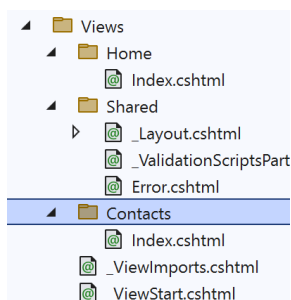


Рисунок 4.3 – Задання ім'я для шаблону представлення

Після виконання цих дій в вікні Solution Explorer з'явиться нова папка contacts з шаблоном Index.



Відкриємо файл Index.cshtml з папки Contacts та додаймо деякий код, наприклад:

```
@model Contact
@{
    ViewData["Title"] = "Зворотній зв'язок";
}

<div>
    65029, м. Одеса, вул. Мечникова, 34
    <br />Приймальна комісія
    <br />Телефон: +38(068)969-86-82
    <br />Пошта: priem@onmu.org.ua
</div>
```

Запустим наш проект та перейдемо за посиланням «Зворотній зв'язок» (рисунок 4.4)

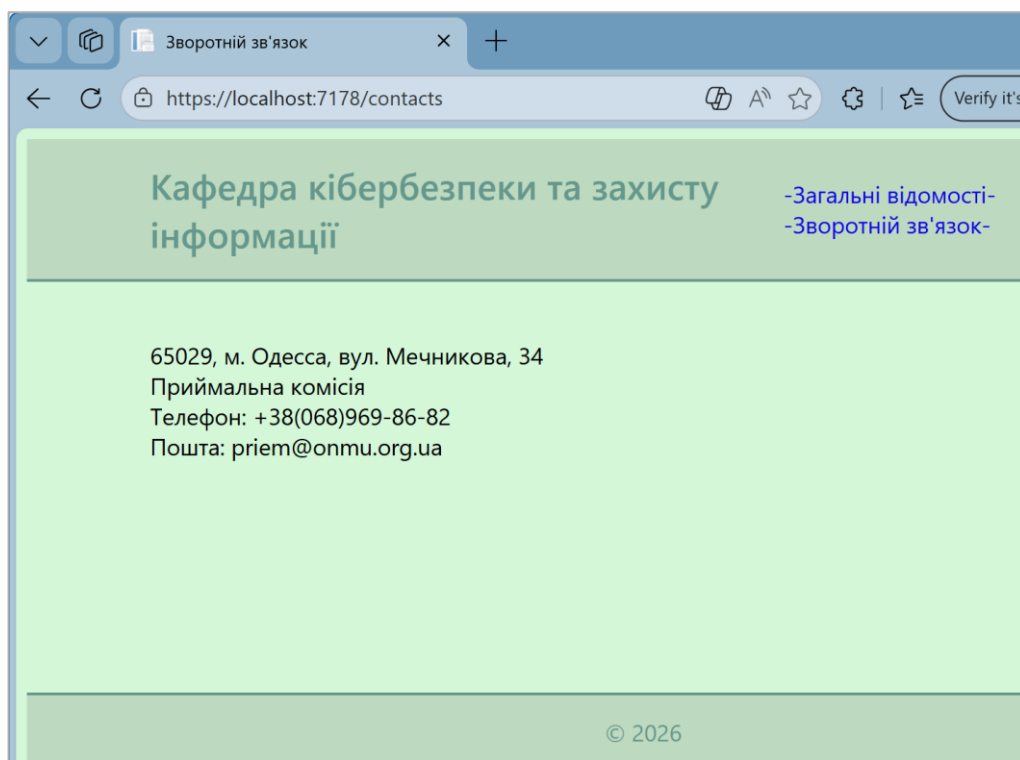


Рисунок 4.4 – Результат додавання нового представлення для сторінки contacts

2. Завдання до практичної роботи 4

За обраною тематикою в створеному веб-додатку передбачити можливість переходу на декілька сторінок (3 – 4 сторінки). Для кожної сторінки створити свій контролер та файл представлення. Заповнити файли представлення інформацією згідно з обраною тематикою.

Практична робота 5. Додавання на сторінку форми

Мета роботи: ознайомитися зі створенням нових моделей. Набути практичних навичок по створенню нових моделей та роботи з формами.

1. Теоретичний матеріал

Розглянемо, як на сторінці «Зворотній зв'язок» додати форму для заповнення інформацією від користувача. За роботу з даними відповідають моделі.

Одним із ключових компонентів патерну MVC є моделі. Ключове завдання моделей – опис структури та логіки даних, що використовуються.

Як правило, всі використовувані сутності у додатку виділяються в окремі моделі, які описують структуру кожної сутності. Залежно від завдань та предметної області, ми можемо виділити різну кількість моделей у додатку.

Усі моделі оформлюються як звичайні класи мовою C#.

Створимо новий клас, тобто модель. Для цього виділяємо папку Models, натискаємо правую кнопку миші, обираємо Add → Class. З'явиться вікно у якому вводимо назву, наприклад Contact.cs.

В середині цього класу напишемо поля, які ми бажаємо мати на формі. Нехай, наприклад, будемо мати такі поля:

```
namespace AspNetCoreMvc.Models
{
    public class Contact
    {
        public string Name { get; set; }

        public string Surname { get; set; }

        public string Email { get; set; }

        public string PhoneNumber { get; set; }

        public string Message { get; set; }
    }
}
```

Відкриємо файл Index.cshtml з папки Views\Contacts. Напишемо, що цей файл буде взаємодіяти з моделлю Contact.

```
@model Contact
@{
    ViewData["Title"] = "Зворотній зв'язок";
}
```

Далі нижче добавимо тег <form>. В якості атрибутів тега вкажемо метод передачі даних, в якому контролері, який метод буде обробляти цю форму:

```
<form method="post" asp-controller="Contacts" asp-action="Check">

</form>
```

Для опису даних будемо використовувати мітку для позначки назви поля, поле для вводу та де буде зберігатися інформація про неправильний ввід:

```
<form method="post" asp-controller="Contacts" asp-action="Check">
  <label asp-for="Name"></label>
  <input asp-for="Name" />
  <span asp-validation-for="Name"></span>
</form>
```

Аналогічно прописуємо для всіх інших полів.
Наприкінці прописуємо вивід кнопки:

```
<input type="submit" value="Відправити" />
```

Також можна додати код в файл site.css. Наприклад:

```
main label{
  width: 100%;
  margin-top: 10px;
  display: block;
}

main input:not([type="submit"]) {
  width: 70%;
  padding: 10px 14px;
  font-size: 14px;
  border-radius: 5px;
  border: 1px solid green;
  outline: none;
  margin-top: 7px;
}

main input:not([type="submit"]):focus {
  border: 3px solid green;
}

main input[type="submit"] {
  width: 40%;
  padding: 10px 0;
  margin-top: 10px;
  text-align: center;
  color: #1002ef;
  background: #b6dbbe;
  font-weight: bold;
  cursor: pointer;
}

main input[type="submit"]:hover {
  background: #0094ff;
}
```

```

main form span {
    color: #ff0000;
    font-weight: bold;
    display: block;
    margin: 5px 0;
}

```

Можна також змінити назву полів при відображенні на екрані. Для цього перейдемо до файлу Contact.cs, підключимо:

```
using System.ComponentModel.DataAnnotations;
```

Перед полями додано атрибут, де напишемо назву полів, що буде відображатися на екрані, наприклад:

```

[Display(Name = "Ваше ім'я")]
public string Name { get; set; }

```

Аналогічно прописуємо для всіх інших полів.
Якщо зараз запустити програму, то будемо мати:

ASP.NET MVC підтримує можливість перевірки, доповнюючи модель об'єкта атрибутами перевірки нотаток до даних. Атрибути перевірки перевіряються на стороні клієнта перед розміщенням значень на сервері, а також на сервері перед виконанням дії контролера.

За допомогою атрибутів валідації моделі ми можемо керувати валідацією та укласти нескладну логіку перевірки значень властивостей вже в атрибути цих властивостей, не вдаючись до коду.

Нижче наведено деякі з вбудованих атрибутів перевірки.

- [Required]: застосування цього атрибуту до властивості моделі означає, що ця властивість повинна бути обов'язково встановлена. За допомогою властивості ErrorMessage цього атрибуту можна налаштувати повідомлення, що виводиться при валідації. А якщо ми явно не встановимо текст повідомлення, то при виведенні помилки буде відображатися стандартний текст повідомлення.
- [RegularExpression]: використання цього атрибуту передбачає, що значення, що вводиться, повинно відповідати вказаному в цьому атрибуті регулярному виразу.
- [StringLength]: щоб користувач не міг ввести дуже довгий текст, застосовується цей атрибут. Першим параметром у конструкторі атрибута йде максимальна допустима довжина рядка. Іменовані параметри, зокрема MinimumLength та ErrorMessage, дозволяють встановити додаткові опції відображення.
- [Compare]: атрибут гарантує, що дві властивості об'єкта моделі мають те саме значення.
- [CreditCard]: перевіряє, чи має кредитний формат карта.
- [EmailAddress]: перевіряє, чи має властивість формат електронної пошти.
- [Phone]: перевіряє, чи має властивість формат телефонного номера.
- [Range]: перевіряє, чи значення властивості вказаний в діапазоні.
- [URL-адреса]: перевіряє, чи має властивість формат URL-адреси.

Повний список атрибутів перевірки можна знайти у [System.ComponentModel.DataAnnotations](#) просторі імен.

Найбільш поширений приклад RegularExpression – це перевірка адреси електронної пошти на коректність. Наприклад,

```
[RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}", ErrorMessage = "Некоректна адреса")]
public string Email { get; set; }
```

Приклад атрибуту StringLength:

```
[StringLength(50, MinimumLength = 3, ErrorMessage = "Довжина рядка має бути від 3 до 50 символів")]
public string Name { get; set; }
```

Приклад атрибуту Range:

```
[Range(1, 110, ErrorMessage = "Неприпустимий вік")]
public int Age { get; set; }
```

Атрибут Compare застосовується, коли потрібно, щоб користувач ввів пароль двічі:

```
[Required]
public string Password { get; set; }

[Compare("Password", ErrorMessage = "Паролі не співпадають")]
public string PasswordConfirm { get; set; }
```

Додаймо валідації (перевірку) до нашої форми. Наприклад, бажано показати які поля є обов'язковими для заповнення:

```
[Display(Name = "Ваше ім'я")]
[Required(ErrorMessage = "Ви забули ввести своє ім'я")]
public string Name { get; set; }
```

Якщо, наприклад, бажаємо ще перевіряти кількість введених символів, то знадобиться атрибут `StringLength`.

```
[Display(Name = "Додайте Ваше повідомлення")]
[Required(ErrorMessage = "Ви забули ввести своє повідомлення")]
[StringLength(20, ErrorMessage = "Необхідно ввести текст щонайменше 20 символів")]
public string Message { get; set; }
```

Створимо метод, який буде приймати введені дані. Перейдемо в клас `ContactsController`, створимо по аналогії з методом, що там маємо, новий метод `Check`. Також за допомогою атрибуту вкажемо що метод `Check` буде приймати данні методом `post`:

```
using ASPNETCORE.Models;
using Microsoft.AspNetCore.Mvc;

namespace ASPNETCORE.Controllers
{
    public class ContactsController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public IActionResult Check(Contact contact)
        {
            return View();
        }
    }
}
```

В середину цього методу додаймо перевірку на правильність вводу та що потрібно зробити або показати, якщо все гаразд, або є помилки.

```
[HttpPost]
public IActionResult Check(Contact contact)
{
    if (ModelState.IsValid)
    {
        return RedirectToAction("Index", "Home");
    }

    return View("Index");
}
```

У подальшому, крім коду `return RedirectToAction("Index", "Home");` можна написати код для зберігання даних у базі даних або їх відправку по пошті.

Запустимо проект, та не заповнимо якесь поле та натиснемо «Відправити». Тоді маємо:

Зворотній зв'язок

https://localhost:7178/Contacts/Check

Кафедра кібербезпеки та захисту інформації

-Загальні відомості-
-Зворотній зв'язок-

Ваше ім'я

Василь

Прізвище

Лісовий

Ваша пошта

mymail@gmail.com

Номер телефону

999-88-77

Додайте Ваше повідомлення

Ви забули ввести своє повідомлення

Відправити

Якщо виправимо всі помилки та знову натиснемо «Відправити», перейдемо до сторінки Home.

Створимо нове представлення, в якому подякуємо користувачеві, що надіслав повідомлення. Для цього в папці View\Contacts створимо новий файл представлення «Answers.cshtml». Додаймо в цей файл, наприклад наступне:

```
@{  
    ViewData["Title"] = "Зворотній зв'язок";  
}  
  
<div>  
    <h1>Дякую @ViewData["Name"] @ViewData["Surname"] що надіслали нам повідомлення!!! </h1>  
    <br />  
</div>
```

До файлу контролера Contacts додаймо новий метод, та змінимо метод Check:

```

[HttpPost]
public IActionResult Check(Contact contact)
{
    if (ModelState.IsValid)
    {
        return RedirectToAction("Answer", contact);
    }

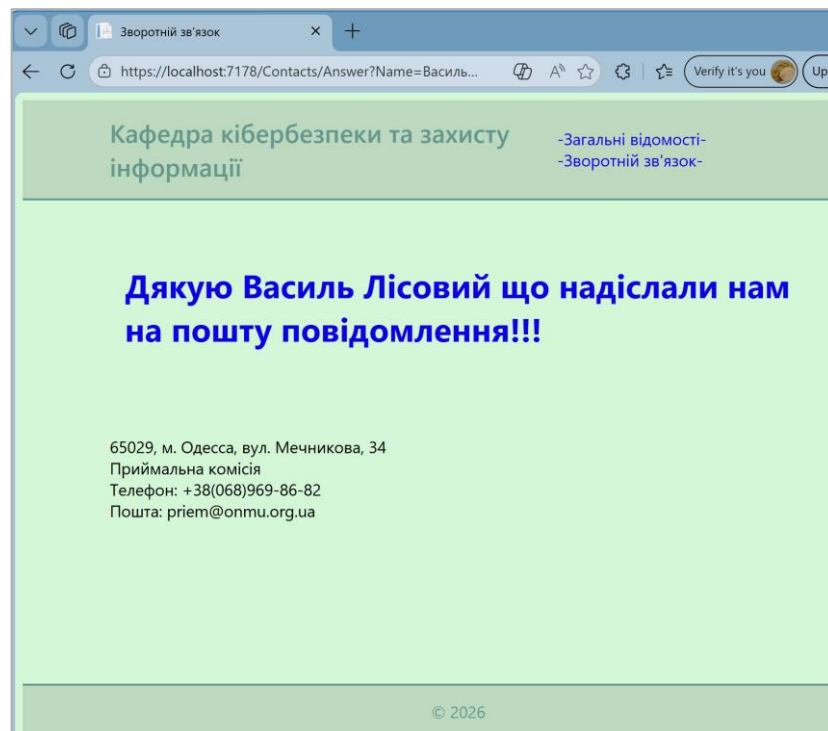
    return View("Index");
}

public IActionResult Answer(Contact contact)
{
    ViewData["Name"] = contact.Name;
    ViewData["Surname"] = contact.Surname;

    return View();
}

```

Якщо заповнимо правильно всі поля та натиснемо «Відправити», перейдемо до сторінки Answers.



2. Завдання до практичної роботи 5

За обраною тематикою в створеному веб-додатку створіть нову сторінку з формою або додайте форму до існуючої сторінки. Створіть модель для роботи з даними форми, використовуйте різні атрибути валідації. Створіть нову сторінку де покажете введені данні.

Література

1. Rafique S. Systematic Review of Web Application Security Vulnerabilities Detection Methods / Sajjad Rafique, Mamoon Humayun, Zartasha Gul, Ansar Abbas, Hasan Javed // Journal of Computer and Communications. - 2015. №3. P. 28-40
https://www.researchgate.net/publication/283180137_Systematic_Review_of_Web_Application_Security_Vulnerabilities_Detection_Methods
2. Clement C. Aladi Web Application Security: A Pragmatic Expose // Digital Threats: Research and Practice. 2024. Volume 5, Issue 2. P. 1 - 9
<https://dl.acm.org/doi/full/10.1145/3644394>
3. Ersahin B. Web application security / Buket Ersahin, Mustafa Ersahin // South Florida Journal of Development, Miami. 2022. v.3, n.4. p.4194-4203.
<https://www.southfloridapublishing.com/ojs/index.php/jdev/article/download/1584/1257>
4. OWASP Top Ten Web Application Security Risks [Електронний ресурс] Режим доступу: https://owasp.org/www-project-top-ten/?spm=a2ty_o01.29997173.0.0.7fcf55fbdllPmU
5. OWASP Foundation. "Developer Guide" [Електронний ресурс] Режим доступу: https://owasp.org/www-project-developer-guide/?spm=a2ty_o01.29997173.0.0.7fcf55fbdllPmU
6. ASP.NET Core security topics [Електронний ресурс] Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-10.0>
7. OWASP ASP.NET Core Security Cheat Sheet [Електронний ресурс] Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/DotNet_Security_Cheat_Sheet.html
8. Анипченко О.Є. Дослідження ступеня захищеності web-додатків на основі аналізу їх структури та інформаційного наповнення / О.Є.Анипченко, Ю.В.Щавінський // Сучасний захист інформації. - №3(51), 2022. - С. 39 - 47.
<https://journals.duikt.edu.ua/index.php/dataprotect/article/download/2648/2546>

Інтернет ресурси

1. Офіційний сайт Visual Studio Community [Електронний ресурс] URL: <https://visualstudio.microsoft.com/ru/vs/community/>