МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: *«**Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів**»*

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Тищук Василь Ігорович

Керівник: ***Красюк Б.В., викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса*
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента

Тищука Василя Ігоровича

1. Тема роботи: ***Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів.***

керівник роботи: Красюк Б.В., викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: аналіз конкурентів на ринку, код серверної та клієнтської частини, git репозиторій проєкту.

4. Перелік завдань, які належить виконати: спроектувати базу даних; підібрати актуальні бібліотеки для застосування під час розробки; здійснити дослідження на наявність конкурентів; реалізувати клієнтську частину застосунку; реалізувати серверну частину застосунку.

5. Перелік графічного матеріалу: відображення досліджень та результатів, отриманих під час написання кваліфікаційної роботи у вигляді рисунків.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Красюк Б.В.	01.12.2023	01.12.2023
2	Красюк Б.В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи	до 31.10.2023 р.	
2	Постановка технічного завдання	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 10.12.2023 р.	
4	Написання розділу 1	до 01.02.2024 р.	
5	Написання розділу 2	до 01.03.2024 р.	
6	Тестування системи	до 20.04.2024 р.	
7	Виправлення помилок	до 01.05.2024 р.	
8	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи/проєкту	до 31.05.2024 р.	
9	Здача кваліфікаційної роботи на кафедру	31.05.2024 р.	

Студент: _____ **Василь ТИЩУК**

Керівник кваліфікаційної роботи: _____ **Богдан КРАСЮК**

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: *Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів.*

Автор: *Тищук Василь Ігорович*

Науковий керівник: *Красюк Б.В., викладач кафедри ЕММІТ*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *73 с., 16 рис., 10 додатків, 28 джерел.*

Ключові слова: *веб-застосунок, серверна частина, клієнтська частина, ASP.NET*

Короткий зміст праці:

Робота присвячена проєктуванню та розробці інтерактивного менеджера сніпетів програмних кодів, який надасть користувачам можливість ефективного управління, зберігання та швидкого доступу до часто використовуваних фрагментів коду під час розробки програмного забезпечення. Актуальність теми зумовлена стрімким розвитком сучасних технологій та зростаючими вимогами до продуктивності розробників. В першому розділі роботи був здійснений аналіз наявних аналогів, що дозволило визначити основні вимоги та сформулювати завдання, необхідні для реалізації проєкту. Також в цьому розділі описуються інструменти використані для організації роботи. В другому розділі описується технологічний стек та кроки необхідні для реалізації застосунку, зокрема проєктування бази даних, розробка серверної частини та клієнтської частини використовуючи фреймворк ASP.NET, Blazor та Tailwind CSS. Робота завершується висновками, які узагальнюють отримані результати та у яких підсумовано виконані кроки необхідні для реалізації застосунку.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Design and development of an interactive program code snippet manager.*

Author: *Tyshchuk Vasyl Ihorovych*

Scientific supervisor: *Krasiuk B.V., Lecturer of the Department of EMMIT*

Defended «……»…… of 2024.

Explanatory note to the qualification work: *73 p., 16 pic., 10 attachments, 28 sources.*

Keywords: *web application, backend, frontend, ASP.NET*

Summary of the paper:

This qualification work is devoted to design and develop an interactive program code snippet manager that will provide users with the ability to effectively manage, store, and quickly access frequently used code snippets during software development. The relevance of the topic is driven by the rapid development of modern technologies and growing demands on developer productivity. The first section of the paper analyzes existing solutions, which allows us to identify the main requirements and formulate the tasks necessary to implement the project. This section also describes the tools used to organize the work. The second section describes the technology stack and the steps required to implement the application, including database design, server side and client side development using the ASP.NET framework, Blazor, and Tailwind CSS. The paper ends with conclusions that summarize the obtained results and all the steps taken to implement the application.

_____

ЗМІСТ

ВСТУП

В нинішньому інформаційному суспільстві, де розвиток програмного забезпечення визначає темпи технологічного прогресу, стає важливою задачею забезпечення зручного та ефективного інструментарію для розробників. Сучасний ритм інновацій та стрімкий розвиток технологій вимагають від фахівців у галузі програмування не тільки високих технічних навичок, але й наявності потужного інструментарію для оптимального втілення своїх ідей. В контексті цього, стає зрозумілою важливість забезпечення розробників інструментарієм, що не лише відображає останні технологічні досягнення, але й є зручним та високоефективним у використанні. Таким чином, створення та вдосконалення інструментів для розробників, зокрема інтерактивних сніпет менеджерів, є актуальною метою.

Обрана тема "Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів" набуває важливості та актуальності в контексті швидкого розвитку сучасних технологій та високих вимог до продуктивності розробників. З усе зростаючою складністю проєктів програмного забезпечення та необхідністю швидкої реалізації ідей, ефективне управління та використання фрагментів коду стає стратегічно важливим завданням. Інтерактивний менеджер сніпетів відповідає цьому виклику, надаючи розробникам зручний інструмент для ефективного зберігання, керування та використання фрагментів коду, що робить його важливим інструментом в арсеналі розробників.

Об'єктом дослідження є ринок програм для управління сніпетами програмного коду. Предметом дослідження є процес проєктування та розробки інтерактивного менеджера сніпетів програмного коду з використанням сучасних інформаційних технологій та інструментів, таких як ASP.NET Core, WebAPI та Blazor WebAssembly.

Мета дослідження полягає у дослідженні теми, проєктуванні та розробці інтерактивного менеджера сніпетів програмних кодів. Для досягнення мети потрібно виконати ряд завдань:

- Здійснити аналіз наявних аналогів;
- Спроектувати систему:

    - Вибір технологій для розробки вебдодатку;
    - Спроектувати базу даних
    - Спроектувати API;
    - Спроектувати інтерфейс користувача (UI).
- Розробити систему згідно плану та протестувати її.

Використовуватимуться на ступні методи дослідження:

- Емпіричні:
    - Експериментальний метод – використовуватиметься для вирішення проблем з додаванням нових технологій, створення власних методів для вирішення проблем локалізації та тестування;
    - Спостереження і опис – використовуватиметься для аналізу конкурентів і роботи власного програмного продукту.
- Теоретичні:
    - Аналіз – робиття всіх питань, пізнання, абстрагування його окремих сторін чи аспектів;
    - Дедукція – виведення висновку;
    - Пояснення та узагальнення;

Отже, ми коротко оглянули потрібність створення менеджера сніпетів програмних кодів. Сформулювали мету проєкту, завдання які потрібно виконати для створення проєкту та список методів для дослідження поставлених завдань і питань.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІНТЕРАКТИВНОГО МЕНЕДЖЕРА СНІПЕТІВ ПРОГРАМНИХ КОДІВ

1.1. Огляд наявних аналогів

Аналіз наявних програмних продуктів у сфері управління сніпетами коду свідчить про те, що, незважаючи на певний рівень функціональності, існуючі рішення мають свої обмеження та недоліки, які можуть ускладнювати повсякденну роботу розробників.

- Багато популярних редакторів коду та інтегрованих середовищ розробки надають інструменти для управління сніпетами, проте їхні можливості обмежені локальним зберіганням. Вони часто не забезпечують зручної синхронізації між різними пристроями чи редакторами, що може призводити до втрати даних або невідповідності між різними пристроями користувача через відсутність ефективної синхронізації.
- Іншим недоліком інтегрованих інструментів є обмежений функціонал для організації сніпетів. Розробники можуть стикатися із відсутністю зручних інструментів для швидкого пошуку, фільтрації та категоризації сніпетів, що ускладнює їхню повсякденну роботу, та може призводити до втрати часу на пошук необхідного фрагменту коду.
- Сервіси з застарілим інтерфейсом, такі як snippleaf.com, можуть виглядати неактуально та не забезпечувати сучасний рівень зручності користування, що може впливати на ефективність роботи розробників та ускладнювати навігацію для користувачів.
- Деякі сайти, наприклад codesnip.net, обмежені лише веб-інтерфейсом, що може бути незручним для розробників, вимагаючи постійного переходу між редакторами та веб-сайтом, що може втручатися у робочий процес.
- Деякі додатки, такі як snippetmanager.net мають лише десктопну версію, що

може обмежити зручність доступу до сніпетів та їхнього управління, зокрема, у випадку використання різних пристроїв.

Обрана тема "Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів" спрямована на вдосконалення і вирішення недоліків існуючих рішень у сфері управління сніпетами. Мета полягає в створенні ефективного та зручного інструменту для розробників, який надасть їм можливість ефективно керувати та використовувати фрагменти коду, враховуючи зазначені проблеми.

1.2 Постановка задачі

Додаток матиме наступні особливості, функції та можливості як:

- Домашня сторінка з короткою інформацією про додаток, яка дозволяє швидко ознайомитися з його основними перевагами.
- Інтуїтивний інтерфейс, який робить використання додатку простим і забезпечує зручність управління.
- Вбудована синхронізація за допомогою зберігання базі даних в хмарі.
- Можливість реєстрації та входу з використання звичних сервісів таких як Google.
- Функіонал для створення, видалення, редагування та пошуку сніпетів.
- Функіонал для швидкого копіювання інформації до буферу обміну.
- Безкоштовне користування додатком, що дає можливість користувачам використовувати всі функції та можливості без необхідності оплати.

Отже, ми описали особливості додатку “Snippy”, також його функції та можливості.

1.3 Організація роботи

Для розроблення веб додатку було обране програмне забезпечення *Visual Studio* [17] та *SQL Server Management Studio* [18]. Visual Studio дозволяє створювати

серверну логіку та клієнтську частину додатку, використовуючи сучасні веб-технології та фреймворки, а SSMS забезпечує ефективне управління базою даних, яка обслуговує веб-додаток.

Visual Studio - це інтегроване середовище розробки (IDE), яке надає розробникам широкий набір інструментів для написання, налагодження та тестування програмного забезпечення.

SQL Server Management Studio (SSMS) - це інструмент для управління базами даних, розроблений для роботи з Microsoft SQL Server. SSMS надає користувачам інтерфейс для адміністрування, конфігурування та управління базами даних, включаючи створення, змінення та видалення баз даних і їх об'єктів, виконання SQL-запитів, моніторинг продуктивності, налаштування безпеки та створення резервних копій.

Щоб спростити і прискорити процес розробки та надати можливість іншим людям ознайомитися з результатами роботи, було прийнято рішення розмістити проєкт на *GitHub* [16].

GitHub - це веб-платформа для зберігання проектів на основі системи контролю версій Git [19], яка забезпечує можливість спільної розробки програмного забезпечення. Вона дозволяє користувачам створювати репозиторії для своїх проектів, завантажувати код, відстежувати його зміни, пропонувати зміни у вигляді пулл-реквестів, вести обговорення та керувати проектом через систему завдань і вікі-сторінки. GitHub також служить центром розробників, надаючи можливість вивчення та використання відкритих проектів, а також співпрацювати з іншими розробниками по всьому світу.

GitHub також надає інтегровані засоби для неперервної інтеграції та постійного розгортання (CI/CD), що спрощує автоматизацію процесу розробки програмного забезпечення. Наприклад, він дозволяє налаштувати збірку та тестування програмного коду при кожному зміні в репозиторії (CI), а також автоматично розгортати нові версії програмного забезпечення в середовища виробництва або тестування (CD). Засоби CI/CD у GitHub можуть бути налаштовані за допомогою спеціальних конфігураційних файлів, таких як GitHub Actions [22] або

інші сторонні інструменти, що дозволяють автоматизувати та оптимізувати робочі процеси.

GitHub набув широкої популярності серед розробників та проектів з відкритим вихідним кодом завдяки своєму зручному інтерфейсу, потужним функціям співпраці та активній підтримці спільноти.

Висновки до розділу 1

Розділ 1 надає огляд наявних аналогів у сфері управління сніпетами коду та виявляє низку обмежень і недоліків існуючих рішень. Обрана тема "Проєктування та розробка інтерактивного менеджера сніпетів програмних кодів" націлена на вдосконалення і вирішення цих недоліків.

Спираючись на результати аналізу конкурентів, було сформульовано чіткий план роботи. План охоплював різні аспекти, зокрема створення гостьової сторінки, яка надавала б стислу інформацію про програму, реалізацію зручного інтерфейсу, а також функціонал для зручного керування сніпетами.

Впроваджуючи ці функції, додаток “Snippy” мав на меті забезпечити безперебійну та ефективну роботу користувачів, а також задовольнити різноманітні потреби та уподобання розробників.

РОЗДІЛ 2
ПРИКЛАДНА ЧАСТИНА РОЗРОБКИ ВЕБ-ДОДАТКУ

2.1. Серверна частина веб-додатку “Snippy”

2.1.1. Стек технологій використаних у розробці серверної частини

Для розроблення веб додатку був обраний фреймворк **ASP.NET Core** [2], який надає розробникам гнучкі та високопродуктивні інструменти для створення сучасних веб-застосунків та сайтів за допомогою платформи .NET [1] та мови програмування C#. ASP.NET Core - це відкритий (Open Source), крос-платформений фреймворк для розробки веб-застосунків, розроблений компанією Microsoft. Ось деякі ключові його переваги:

1. Крос-платформенність: ASP.NET Core є мультиплатформеним фреймворком, що дозволяє розгортати веб-застосунки на різних операційних системах, включаючи Windows, Linux, Docker та macOS. Це забезпечує високу гнучкість у виборі середовища для розробки та експлуатації веб-додатків.
2. Модульність та гнучкість: Фреймворк побудований з урахуванням модульності, що дозволяє вам вибирати та використовувати тільки ті частини, які необхідні для вашого застосунку. Це спрощує розробку та розгортання.
3. Інтеграція з Entity Framework [7] та LINQ [20]: ASP.NET Core гарантує ефективну роботу з базами даних завдяки інтеграції з Entity Framework, який дозволяє розробникам працювати з базами даних, використовуючи об'єктно-реляційне відображення та мову запитів LINQ.
4. Висока Швидкість та Продуктивність: ASP.NET Core оптимізований для високої продуктивності та ефективності. Він має вбудовану підтримку асинхронного програмування, яка дозволяє обробляти більше запитів за один час і поліпшує відгук застосунку.

5. Оптимізована для хмарних сервісів: Фреймворк має вбудовану підтримку для розгортання в хмару, що робить його ідеальним вибором для розробників, які планують використовувати хмарні сервіси для своїх застосунків.
6. Вбудована безпека: Забезпечує різноманітні механізми безпеки, включаючи автентифікацію та авторизацію, що допомагає захищати застосунок від різних загроз безпеки.

Для розробки серверної частин (API) був використаний ASP.NET Core **Web API** [4] - це частина ASP.NET Core, призначена для створення та розгортання веб-служб на платформі .NET Core. Web API дозволяє розробникам створювати легкі, швидкі та ефективні веб-служби, які можуть бути використані для взаємодії та обміну даними між різними системами, клієнтами, такими як веб-додатки, мобільні/десктопні додатки, плагіни або інші служби.

Ось його ключові аспекти:

1. RESTful архітектура: Web API підтримує архітектурний стиль REST (Representational State Transfer), що дозволяє взаємодіяти з ресурсами за допомогою стандартних HTTP-методів, таких як GET, POST, PUT та DELETE. Це забезпечує простоту, легкість розуміння та взаємодію між різними складовими системи.
2. Маршрутизація: WebAPI використовує атрибути маршрутизації, що дозволяють визначити, як HTTP-запити відображаються на методи контролера. Контролери у WebAPI відповідають за обробку запитів та визначення логіки для взаємодії з даними.
3. Обмін Даними: WebAPI використовує формати обміну даними, такі як JSON або XML, для передачі інформації між клієнтом та сервером. Це робить дані легкими для розуміння, а також забезпечує простоту інтеграції та взаємодії з іншими системами, гнучкість та масштабованість.
4. Фільтри та атрибути: Web API використовує систему фільтрів та атрибутів для додаткового контролю над обробкою HTTP-запитів та відповідей.

5. Вбудована підтримка аутентифікації та авторизації: Для забезпечення безпеки Web API має вбудовані засоби для реалізації аутентифікації та авторизації, що дозволяє обмежувати доступ до ресурсів.
6. Підтримка Асинхронності: Здатність обробляти запити асинхронно є однією з ключових переваг WebAPI. Це дозволяє оптимізувати ресурси сервера та забезпечує швидке відгуку служб.
7. Засоби тестування: ASP.NET Core Web API легко тестується, і фреймворк надає зручні інструменти для написання та виконання автоматизованих тестів.

ASP.NET Core Web API є потужним інструментом для створення розподілених веб-служб і використовується для побудови масштабованих та надійних систем.

За допомогою **Entity Framework** [7] була створена база даних. Entity Framework Core (EF Core) - це легкий, ефективний та крос-платформений ORM (Object-Relational Mapping) фреймворк, розробленим для роботи з базами даних у середовищі .NET Core та .NET 5+. ORM дозволяє взаємодіяти з базою даних, використовуючи об'єктно-орієнтовані об'єкти, замість прямих SQL-запитів.

Entity Framework Core є популярним вибором для .NET-розробників, які працюють з базами даних та бажають використовувати об'єктно-реляційний підхід для доступу до даних.

Ось кілька ключових характеристик Entity Framework Core:

1. Об'єктно-реляційне відображення: EF Core надає можливість взаємодіяти з базою даних через об'єктно-реляційне відображення (ORM). Це означає, що розробники можуть працювати з об'єктами в коді, а EF Core автоматично вирішує перетворення цих об'єктів на таблиці в базі даних та навпаки.
2. Підтримка багатьох провайдерів баз даних: EF Core підтримує різні провайдери баз даних, включаючи SQL Server, SQLite, PostgreSQL, MySQL та інші. Це робить його універсальним і придатним для використання в різних проектах. Під час розробки була використана база даних MS SQL Server.

3. Крос-платформенність: Entity Framework Core підтримує роботу на різних платформах, включаючи Windows, Linux та macOS. Це забезпечує гнучкість при розробці та розгортанні додатків.
4. Міграції баз даних: EF Core дозволяє легко керувати змінами схеми бази даних за допомогою механізму міграцій. Можна визначати зміни у моделі об'єктів та автоматично генерувати SQL-скрипти для внесення цих змін в базу даних.
5. LINQ (Language Integrated Query) підтримка: EF Core інтегрується з LINQ, що дозволяє використовувати мовні конструкції C# для написання запитів до бази даних. Це робить код більш зрозумілим та покращує його підтримку та перевірку на етапі компіляції.
6. Вбудована підтримка асинхронності: EF Core надає асинхронні методи для взаємодії з базою даних, що поліпшує продуктивність та масштабованість додатків.
7. Інтеграція з ASP.NET та .NET Core: EF Core легко інтегрується з ASP.NET та .NET Core додатками, що спрощує роботу з базою даних в контексті веб-розробки.
8. Вбудовані функції валідації та відстеження об'єктів: EF Core включає в себе засоби для валідації даних перед взаємодією з базою даних та відстеження змін у стані об'єктів.

В ролі бази даних був використаний **Microsoft SQL Server** [21] - надійна та масштабована система управління реляційними базами даних, розроблена корпорацією Microsoft. Вона надає розширені можливості для управління, зберігання та обробки даних у багатьох компаніях та організаціях різних розмірів і напрямків. До ключових переваг входить:

1. Висока продуктивність: SQL Server відомий своєю швидкістю та ефективністю обробки запитів, що дозволяє швидко виконувати складні операції з великими обсягами даних.
2. Універсальність: Підтримує широкий спектр типів даних та надає потужні можливості маніпулювання даними.

3. Розширюваність: SQL Server легко масштабується, що дозволяє працювати з різними обсягами даних - від невеликих баз даних для невеликих підприємств до великих підприємств з великими обсягами даних.
4. Надійність та стабільність: SQL Server відомий своєю надійністю та стабільністю. Він має вбудовані механізми для забезпечення цілісності даних, резервного копіювання та відновлення, а також моніторингу та управління системою.
5. Широка підтримка інтеграції: SQL Server інтегрується з іншими продуктами Microsoft, такими як Microsoft Azure, а також постачається з різноманітними інструментами для розробки, управління та моніторингу баз даних, такими як SQL Server Management Studio. Крім цього підтримує різні стандарти і протоколи для інтеграції з іншими системами.

Для аутентифікації та авторизації був використаний популярний механізм під назвою **JWT** (JSON Web Token Authentication) [10], який використовує JSON Web Tokens (JWT) для передачі інформації про ідентифікацію між сторонами в безпечний та ефективний спосіб. JWT Authentication забезпечує аутентифікацію та авторизацію між клієнтом та сервером у розподіленому середовищі, такому як веб-додатки чи мікросервісна архітектура.

JWT є відкритим стандартом (RFC 7519), який визначає компактний та самостійний спосіб представлення інформації між двома сторонами у вигляді JSON об'єкта. У випадку аутентифікації, JWT використовується для підтвердження того, що користувач має право доступу до ресурсу.

Основні компоненти JWT:

1. Header (Заголовок): Містить метадані токена та інформацію про тип та алгоритм підпису.
2. Payload (Навантаження): Містить корисну інформацію про сутність (наприклад, користувача) та додаткові дані, які передається між сторонами.

3. Signature (Підпис): Це підпис, який генерується на основі заголовка та навантаження за допомогою ключа. Він використовується для перевірки, чи токен не був змінений під час передачі.

JWT Authentication використовується для двох основних цілей - аутентифікації та авторизації. Під час автентифікації користувач отримує токен після введення правильних облікових даних. Під час авторизації сервер перевіряє токен та визначає, чи користувач має доступ до певного ресурсу чи дії.

Після успішної аутентифікації, сервер генерує JWT та передає його клієнту. Клієнт може зберігати цей токен, наприклад, у куках чи локальному сховищі. Кожен наступний запит від клієнта до сервера включатиме цей токен у заголовку (зазвичай "Authorization"). Сервер перевіряє цей токен та визначає, чи користувач має доступ до запитуваного ресурсу.

Основні переваги використання JWT для аутентифікації включають компактність, простоту та можливість передавати додаткові корисні дані разом з токеном. Однак важливо забезпечити безпеку ключа підпису, оскільки він використовується для перевірки валідності токена.

2.1.2. Створення проєкту

Для розгортання та розробки серверної частини було використано інтегроване середовище розробки Visual Studio [17]. Під час створення нового проєкту виберемо шаблон Blazor WebAssembly Standalone App та натиснемо далі. Після цього вкажемо назву проєкту та шлях де проєкт буде створено. В додаткових параметрах виберемо версію фреймворку та тип аутентифікації. Також відмітимо опцію використання Progressive Web Application [28] яка надасть можливість використовувати веб-додаток як десктопний додаток.

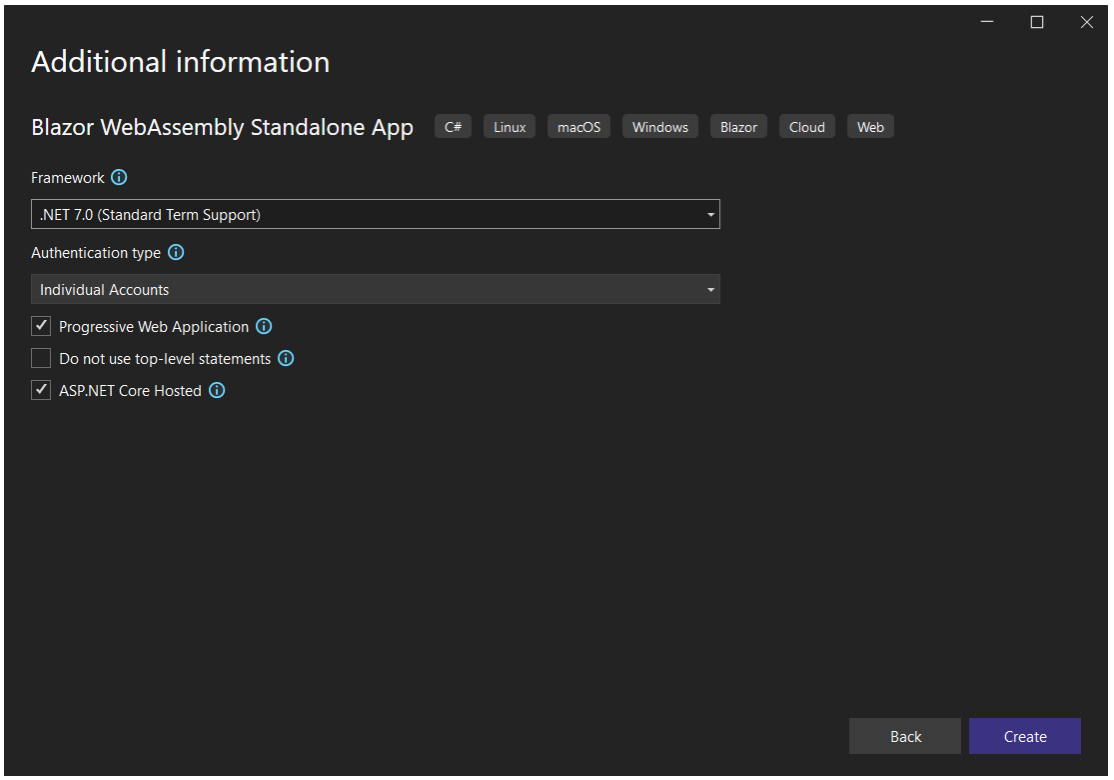

Рис. 2.1. Діалогове вікно “Additional information” під час створення проєкту

Джерело: створено автором

В результаті отримаємо наступну структуру проєкту розділену на клієнтську частину, серверну частину та проєкт для зберігання коду спільного для проєктів Client та Server (наприклад, моделей даних).

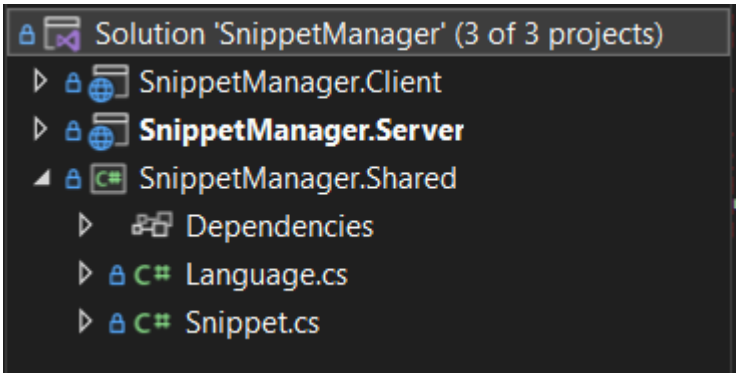

Рис. 2.2. Початкова структура проєкту

Джерело: створено автором

2.1.3. Структура серверної частини

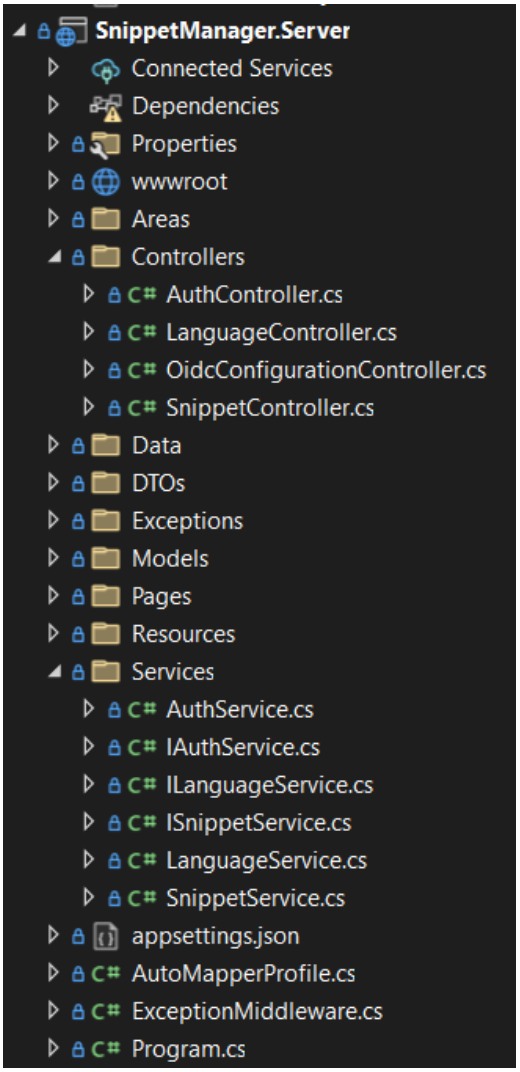

Рис. 2.3. Структура проєкту серверної частини

Джерело: створено автором

Основними директоріями серверної частини є:

Controllers - Директорія Controllers містить контролери, які обробляють HTTP-запити та повертають HTTP-відповіді. Кожен контролер зазвичай відповідає за обробку запитів до певного ресурсу або групи ресурсів і містить методи дії (action methods), які відповідають різним HTTP-методам (GET, POST, PUT, DELETE).

Data - Директорія Data містить класи, пов’язані з доступом до даних, зокрема контексти баз даних (DbContext) та класи для міграцій (Migrations).

DTOs - Директорія DTOs містить об’єкти передачі даних (Data Transfer Objects), які використовуються для обміну даними між шаром контролерів та іншими шарами застосунку або зовнішніми системами. DTO допомагають уникати надмірної передачі даних та забезпечують абстракцію від внутрішньої структури моделі.

Models - Директорія Models містить класи моделей, які представляють дані додатку та відображають структуру бази даних.

Services - Директорія Services містить класи сервісів, які реалізують бізнес-логіку застосунку. Сервіси використовуються для організації коду, що виконує різні завдання, які можуть бути використані повторно в різних частинах застосунку, такі як робота з базами даних, обробка даних, виклики зовнішніх API тощо.

2.1.4. База даних та моделі

Під час проєктування бази даних була визначена структура спрямована на зберігання ключової інформації, зокрема назви сніпета, його вмісту та мови програмування. Також передбачено зберігання даних про користувача, в тому числі ім'я користувача, email, та хешованого паролю для забезпечення безпеки.

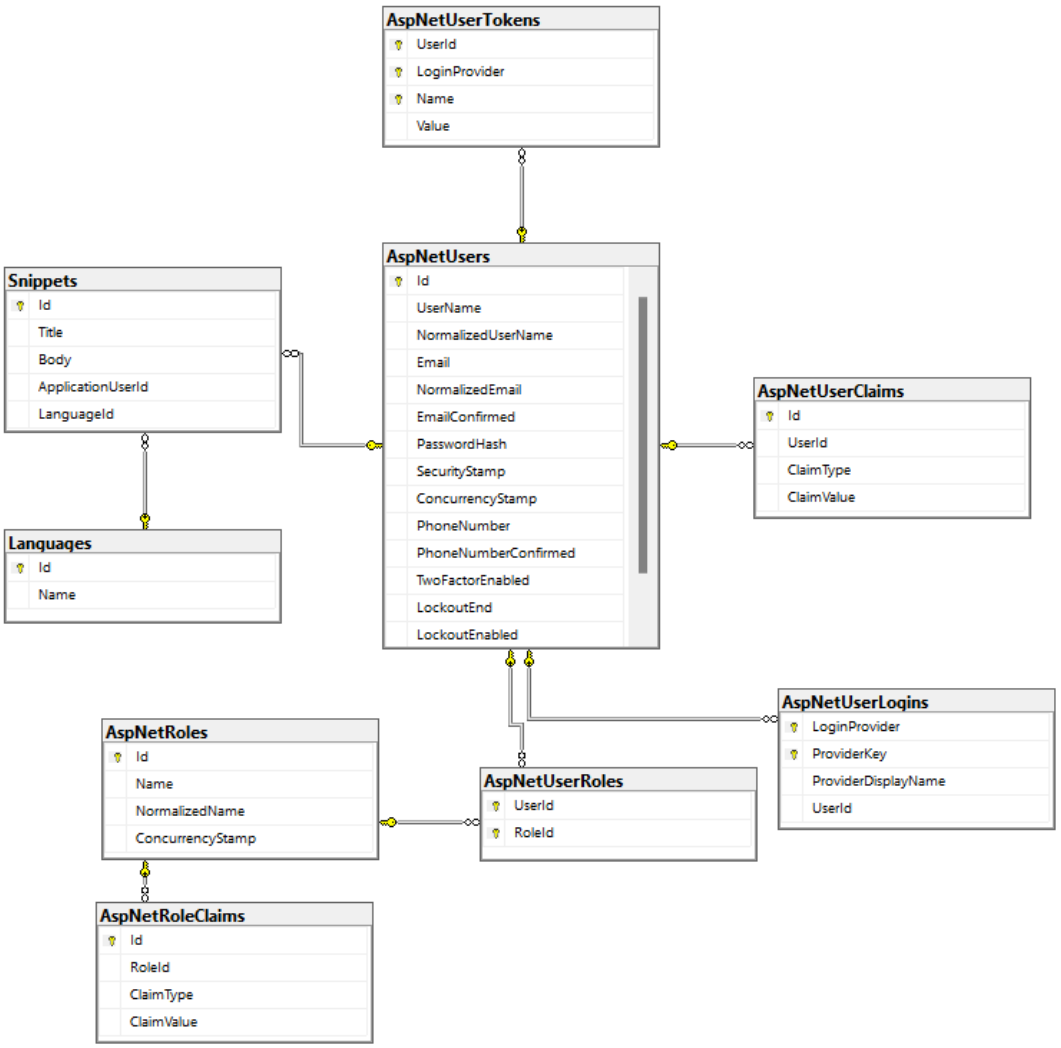

Рис. 2.4. Структура бази даних

Джерело: створено автором

Створена сутність сніпета та мови програмування за допомогою Entity Framework виглядає наступним чином:

Лістинг 2.1. Класс Snippet

```
public class Snippet
{
    public int Id { get; set; }
    public string? Title { get; set; }
    public string? Body { get; set; }
    public string? ApplicationUserId { get; set; }
    ApplicationUser? ApplicationUser { get; set; }
}
```

Лістинг 2.2. Класс Language

```
public class Language
{
	[Key]
	[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
	public int Id { get; set; }
	public string Name { get; set; }
	public virtual ICollection<Snippet>? Snippets { get; set; }
}
```

Також було створено декілька користувачів та тестові дані для наповнення бази даних інформацією, використовуючи метод OnModelCreating() який викликається в класі ApplicationDbContext():

Лістинг 2.3. Seeder бази даних

```
	protected override void OnModelCreating(ModelBuilder modelBuilder)
	{
		base.OnModelCreating(modelBuilder);

		string userId1 = Guid.NewGuid().ToString();
		string userId2 = Guid.NewGuid().ToString();

		var user1 = new ApplicationUser
		{
			Id = userId1,
			UserName = "user@gmail.com",
			Email = "user@gmail.com",
			EmailConfirmed = true,
			NormalizedEmail = "user@gmail.com".ToUpper(),
			NormalizedUserName = "user@gmail.com".ToUpper()
		};
		var user2 = new ApplicationUser
		{
			// ...
		};

		PasswordHasher<ApplicationUser> hasher = new
PasswordHasher<ApplicationUser>();
		user1.PasswordHash = hasher.HashPassword(user1, "user$Pass1");
		user2.PasswordHash = hasher.HashPassword(user2, "user$Pass2");
		modelBuilder.Entity<ApplicationUser>().HasData(user1, user2);
```

```
        modelBuilder.Entity<Snippet>().HasData(
              new Snippet
              {
                    Id = 1,
                    Title = "Hex to RGB in Python",
                    Body = "Lorem ipsum dolor sit amet...",
                    ApplicationUserId = userId1,
              },
              // ...
    }
```

Після цього були створена міграція за допомогою команди Add-Migration та оновлена база даних за допомогою команди Update-Database.

2.1.5. Реалізація серверної частини

Для зручного тестування та використання API була додана підтримка Swagger. Для цього був встановлений NuGet пакунок “Swashbuckle.AspNetCore” та в Program.cs додано наступне:

Лістинг 2.4. Ключові рядки коду відповідальні за Swagger в Program.cs

```
var builder = WebApplication.CreateBuilder(args);
// ...
builder.Services.AddSwaggerGen();
// ...
if (app.Environment.IsDevelopment())
{
     app.UseSwagger();
     app.UseSwaggerUI();
}
```

Функціонал для логіну та реєстрації був реалізований за допомогою JWT токенів. Для цього в Program.cs був доданий JwtBearer сервіс:

Лістинг 2.5. Вміст коду AddJwtBearer

```
builder.Services
.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddJwtBearer(options =>
{
	options.TokenValidationParameters = new TokenValidationParameters
	{
		ValidateIssuer = true,
		ValidateAudience = true,
		RequireExpirationTime = true,
		ValidateIssuerSigningKey = true,
		ValidIssuer =
builder.Configuration.GetSection("Jwt:Issuer").Value,
		ValidAudience =
builder.Configuration.GetSection("Jwt:Audience").Value,
		IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration.GetSecti
on("Jwt:Key").Value))
	};
});
```

Після цього був створений створений контроллер аутентифікації:

Лістинг 2.6. Контроллер аутентифікації

```
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
	private readonly IAuthService _authService;

	public AuthController(IAuthService authService)
	{
		_authService = authService;
	}

	[HttpPost("register")]
	public async Task<IActionResult> Register([FromBody] LoginUser user)
	{
		await _authService.Register(user);
		return Ok();
	}
	[HttpPost("login")]
```

```
	public async Task<IActionResult> Login([FromBody] LoginUser user)
	{
		var token = await _authService.Login(user);
		return Ok(token);
	}
}
```

Для обробки логіки аутентифікації був створений AuthService та в appsettings.json задано JWT Key, Issuer та Audience. Для Google аутентифікації був створений API ключ на сайті console.cloud.google.com та заданий в Program.cs:

Лістинг 2.7. Додавання сервісу Google

```
builder.Services
	.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
	.AddIdentityServerJwt()
	.AddGoogle(googleOptions =>
	{
	googleOptions.ClientId =
builder.Configuration["Authentication:Google:ClientId"];
	googleOptions.ClientSecret =
builder.Configuration["Authentication:Google:ClientSecret"];
	})
```

Для реалізації бекенд частини були створені наступні ендпоінти для клієнтської частини:

1. [POST] api/Auth/login - вхід юзера в систему
2. [POST] api/Auth/register - реєстрування юзера на сайті
3. [GET] /api/Snippet - отримання сніпетів залогіненного користувача
4. [POST] /api/Snippet - створення сніпета
5. [GET] /api/Snippet/{id} - отримання детальної інформації про конкретний сніпет
6. [PUT] /api/Snippet/{id} - оновлення сніпета
7. [DELETE] /api/Snippet/{id} - видалення сніпета
8. [GET] /api/Snippet/search - пошук сніпетів
9. [GET] api/Language - отримання списку доступних мов для підсвітки синтаксису

Для ендпоінтів маніпуляції сніпетами був створений відповідний контроллер (див. ДОДАТОК Л) та сервіс (див. додаток В) в якому реалізована основна логіка

роботи з сніпетами, а саме їх створення, оновлення, видалення та пошук в базі даних.

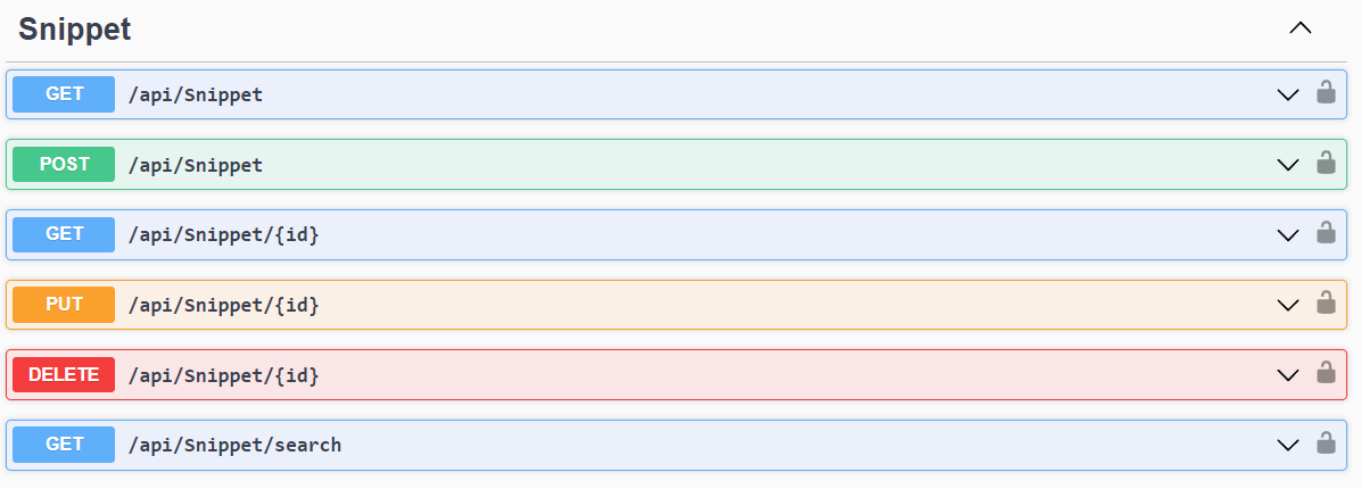

Рис. 2.5. Endpoints маніпуляції сніпетами

Джерело: створено автором

Для ендпоінту отримання списку доступних мов програмування для підсвітки синтаксису був створений подібний контролер та сервіс:

Лістинг 2.8. Контролер для маніпулювання списком мов програмування

```
[Route("api/[controller]")]
[ApiController]
public class LanguageController : ControllerBase
{
	private readonly ILanguageService _languageService;
	public LanguageController(ILanguageService languageService)
	{
		_languageService = languageService;
	}

	[HttpGet]
	public async Task<List<LanguageDto>> GetLanguages()
	{
		return await _languageService.GetLanguages();
	}
}
```

Лістинг 2.9. Сервіс для маніпулювання списком мов програмування

```
public class LanguageService : ILanguageService
	{
	private readonly ApplicationDbContext _context;
	private readonly IMapper _mapper;

	public LanguageService(ApplicationDbContext context, IMapper mapper)
	{
		_context = context;
		_mapper = mapper;
	}

	public async Task<List<LanguageDto>> GetLanguages()
	{
		var languages = await _context.Languages.ToListAsync();
		return _mapper.Map<List<LanguageDto>>(languages);
	}
}
```

В даному прикладі для представлення списку мов програмування замість моделі Language був використаний об’єкт передачі даних LanguageDto створений за допомогою AutoMapper [8] який допомагає уникнути надмірної передачі даних:

Лістинг 2.10. Data Transfer Object для моделі Language

```
public class LanguageDto
{
	public int Id { get; set; }
	public string Name { get; set; }
}
```

Подібним чином був створений DTO для сутності Snippet:

Лістинг 2.11. Data Transfer Object для моделі Snippet

```
public class SnippetDto
{
	public string? Title { get; set; }
	public string? Body { get; set; }
	public int? LanguageId { get; set; }
}
```

2.2. Клієнтська частина веб-додатку “Snippy”

2.2.1 Стек технологій використаних у розробці фронтенду

Для розробки клієнтської частини був використаний **Blazor** WebAssembly [3]. Це технологія, яка дозволяє розробляти веб-застосунки з використанням мови програмування C# та виконувати її безпосередньо в браузері. Blazor WebAssembly є частиною фреймворку ASP.NET Core і дозволяє розробникам використовувати знайомий стек технологій для побудови клієнтської частини веб-застосунків.

Основні характеристики Blazor WebAssembly:

1. Один мовний стек: Однією з ключових особливостей Blazor WebAssembly є можливість використовувати мову програмування C# для розробки клієнтської частини веб-застосунків. Це дозволяє розробникам використовувати ті ж мовні конструкції та інструменти, які вони використовують для розробки серверних застосунків на платформі .NET та ASP.NET Core. Це означає, що розробники можуть використовувати бібліотеки .NET Standard та сервіси ASP.NET Core у своєму клієнтському коді.
2. Використання Razor-синтаксису: Blazor використовує Razor-синтаксис, що дозволяє вставляти C# код безпосередньо в HTML. Це полегшує розробку та утримання клієнтських компонентів.
3. Компонентна модель: Blazor розглядає веб-застосунок як набір взаємодіючих компонентів. Компоненти можуть містити логіку, розмітку та код у C#, бути створені та використані повторно, що спрощує організацію та управління клієнтським кодом.
4. Робота в режимі offline: Blazor WebAssembly може працювати в режимі offline, що означає, що веб-застосунок може функціонувати навіть без постійного підключення до Інтернету. Ресурси та логіка можуть керуватись для подальшого використання в офлайн-режимі.

5. Використання WebAssembly: WebAssembly (Wasm) - це виконавча середа, яка дозволяє виконувати високопродуктивний байт-код у веб-браузері. Blazor WebAssembly використовує WebAssembly для виконання коду, написаного мовою C#, що відкриває нові можливості для розробки веб-додатків.
6. Інтеграція з Visual Studio та іншими інструментами: Blazor WebAssembly інтегрується з середовищем розробки Visual Studio, що полегшує розробку та налагодження клієнтських застосунків.

Blazor WebAssembly надає потужний інструмент для розробки веб-додатків, використовуючи технології .NET та WebAssembly, що робить його привабливим вибором для розробників, які шукають ефективні та сучасні підходи до веб-розробки.

Для створення веб сторінок були використані **Razor Pages** [6]. Вони є важливою частиною фреймворку ASP.NET Core, яка надає простий та ефективний спосіб для розробки веб-сторінок та компонентів.

Основні риси та характеристики Razor Pages:

1. Модель програмування на основі сторінок: У Razor Pages кожна веб-сторінка представляє собою окремий файл з розширенням .razor, який містить HTML-розмітку та вбудований код C#. Це полегшує організацію та розробку веб-сторінок.
2. Мінімальна конфігурація: Однією з переваг Razor Pages є відсутність необхідності в конфігурації контролерів та маршрутів. Кожна сторінка автоматично знаходиться за визначеним маршрутом та співвідноситься зі своїм .razor файлом.
3. Модель обробки подій (Event Handler Model): Razor Pages використовує модель обробки подій, де код обробника подій (event handler) пов'язується з конкретними подіями сторінки. Наприклад, можна визначити метод OnGet для обробки HTTP GET-запитів.
4. Спільне використання коду та розмітки: В Razor Pages можна використовувати спеціальні теги та атрибути для об'єднання коду та HTML. Наприклад, тег

<form> може містити атрибут asp-page-handler, який вказує на метод, який буде викликано при відправці форми.

5. Підтримка HTTP Запитів: Razor Pages підтримують обробку HTTP запитів через атрибути, що дозволяє легко робити HTTP GET, POST, та інші запити.
6. Вбудовані моделі: В Razor Pages можна використовувати вбудовані моделі, які автоматично зв'язуються з введеними даними на сторінці. Моделі можуть визначатися як класи C# та автоматично зчитувати та зберігати дані з HTTP-запитів.

Для стилізації веб-інтерфейсу був використаний **Tailwind CSS** [13] - сучасний фреймворк для створення веб-інтерфейсів, який базується на концепції utility-first та дозволяє замість написання власних CSS-стилів використовувати готові класи для стилізації елементів. Ось деякі його особливості та переваги:

1. Utility-first CSS: Tailwind CSS надає велику кількість класів, кожен з яких представляє конкретну CSS-властивість. Наприклад, .bg-blue-500 задає колір фону в блакитний.
2. Масштабованість та гнучкість: Завдяки використанню готових класів, можна швидко створювати та редагувати веб-інтерфейс без потреби в написанні або зміні власних CSS-стилів.
3. Реактивність та адаптація до мобільних девайсів: Tailwind пропонує можливості для створення реактивних дизайнів за допомогою вбудованих класів для розміщення, вирівнювання і відображення на різних екранах.
4. Розширюваність: Надається можливість легко розширювати фреймворк, додаючи власні класи або налаштовуючи конфігурацію.Це включає вибір кольорів, розмірів, відступів тощо.
5. Оптимізація розміру файлів: Використання лише тих класів, які потрібні для кожного конкретного елементу, дозволяє уникнути зайвого CSS-коду та зменшити розмір вихідних файлів.

2.2.2. Структура клієнтської частини

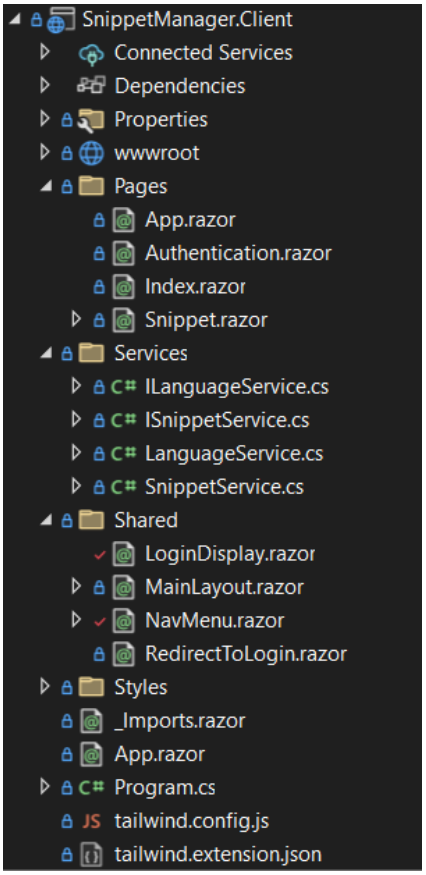

Рис. 2.6. Структура клієнтської частини

Джерело: створено автором

Основними директоріями клієнтської частини є:

wwwroot: Директорія wwwroot використовується для статичних файлів, таких як CSS стилів, зображень та JavaScript скриптів.

Pages: Директорія Pages містить компоненти Razor, які представляють собою сторінки Blazor додатка.

Services: Директорія Services використовується для розміщення сервісів, які надають функціональність для компонентів та використовуються для виклику API ендпоінтів.

Shared: Директорія Shared містить компоненти, які можуть бути спільно використовувані різними сторінками. Наприклад, навігаційне меню та головний шаблон який визначає загальне оформлення сторінок.

2.2.3. Реалізація клієнтської частини

На початковому етапі розробки MVP (Мінімально Життєздатного Продукту) був використаний Bootstrap - фреймворк для розробки веб-додатків і веб-сайтів який використовується в ASP.NET проєктах за замовчуванням та дозволяє швидко створювати стильні та адаптивні веб-додатки з мінімальними зусиллями.

Для виклику API ендпоінтів створимо відповідні сервіси - SnippetService та LanguageService (див. додаток А та Б). Після цього ці сервіси зареєструємо в Program.cs за допомогою Dependency Injection:

Лістинг 2.12. Реєстрація SnippetService та LanguageService в Program.cs

```
...
builder.Services.AddScoped(sp =>
sp.GetRequiredService<IHttpClientFactory>().CreateClient("SnippetManager.S
erverAPI"));
builder.Services.AddScoped<ISnippetService, SnippetService>();
builder.Services.AddTransient<ILanguageService, LanguageService>();
```

Наступним, була створена домашня сторінка користувача та сторінка сніпету за допомогою Razor Pages, повний код якої можна переглянути в додатку Д та Е. Домашня сторінка веб сайту була реалізована наступним чином з використанням HTML та CSS:

Лістинг 2.13. Домашня сторінка веб сайту - Index.razor

```
@page "/"
@using Microsoft.AspNetCore.Authorization;
@inject IJSRuntime JSRuntime
@inject ISnippetService SnippetService
@inject NavigationManager NavigationManager
@inherits LayoutComponentBase

  <style>
      .content {
      display: flex;
      justify-content: center;
      align-items: center;
      text-align: center;
```

```
      flex-direction: column;
      min-height: 95vh;
      padding: 0 30%;
      background-image: linear-gradient(330deg, rgb(190, 199, 243),
rgb(255, 255, 255));
      background-size: cover;
      background-position: center;
      }

      .content p {
      font-size: 1.5em;
      padding: 20px;
      }
  </style>

<div class="content">
      <h1>Snippy</h1>
      <p>Snippet manager for developers designed to effortlessly create and
organize your own personal snippet collection and have quick access to
it</p>
      <a href="/app" class="btn btn-primary btn-lg">Get Started</a>
</div>
```

Для тимчасової зміни іконки копіювання для індикації того що операція виконалась успішно створимо наступну JavaScript функцію:

Лістинг 2.14. JavaScript функція за шляхом wwwroot\js\custom.js

```
window.changeCopyIconTemporary = function (snippetId) {
      var icon = document.getElementById("copy-icon-" + snippetId);
      if (icon) {
      icon.className = "fa-solid fa-check";
      setTimeout(function () {
            icon.className = "fa-regular fa-clipboard";
      }, 1000); // Change back after 1000 milliseconds (1 second)
      }
}
```

В результаті написання сторінок за допомогою Razor Pages та Bootstrap веб сайт мав наступний вигляд:

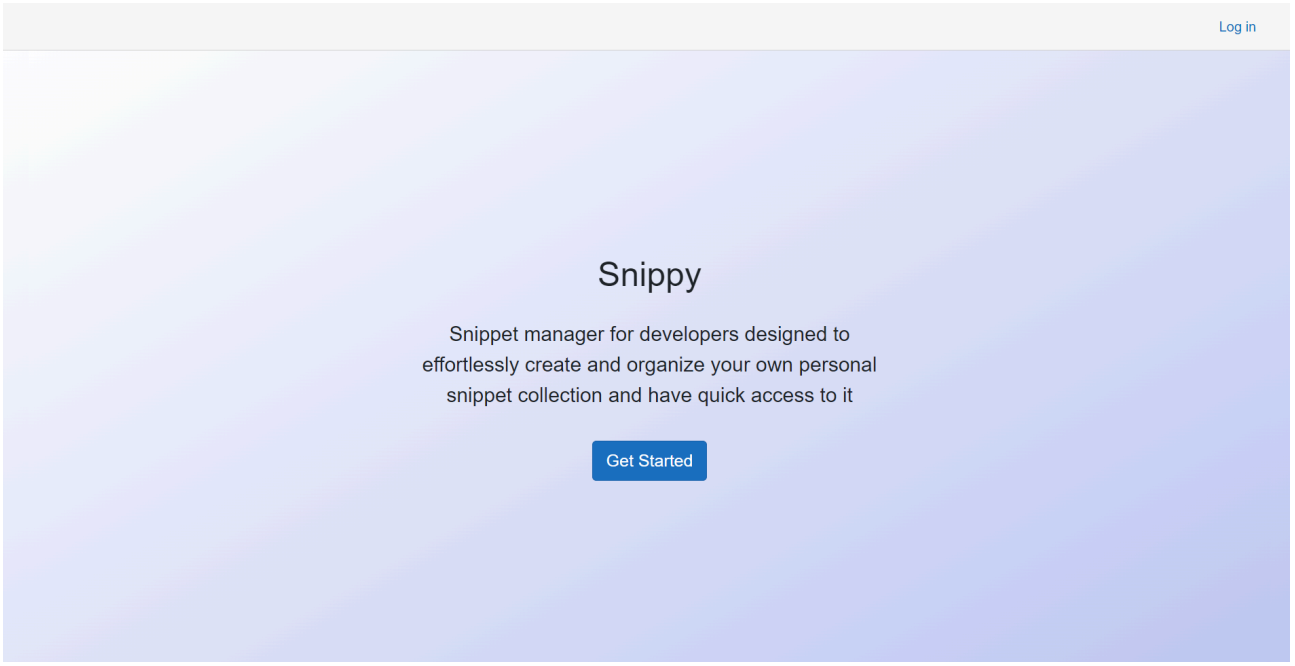

Рис. 2.7. Вигляд домашньої сторінки вебсайту

Джерело: створено автором

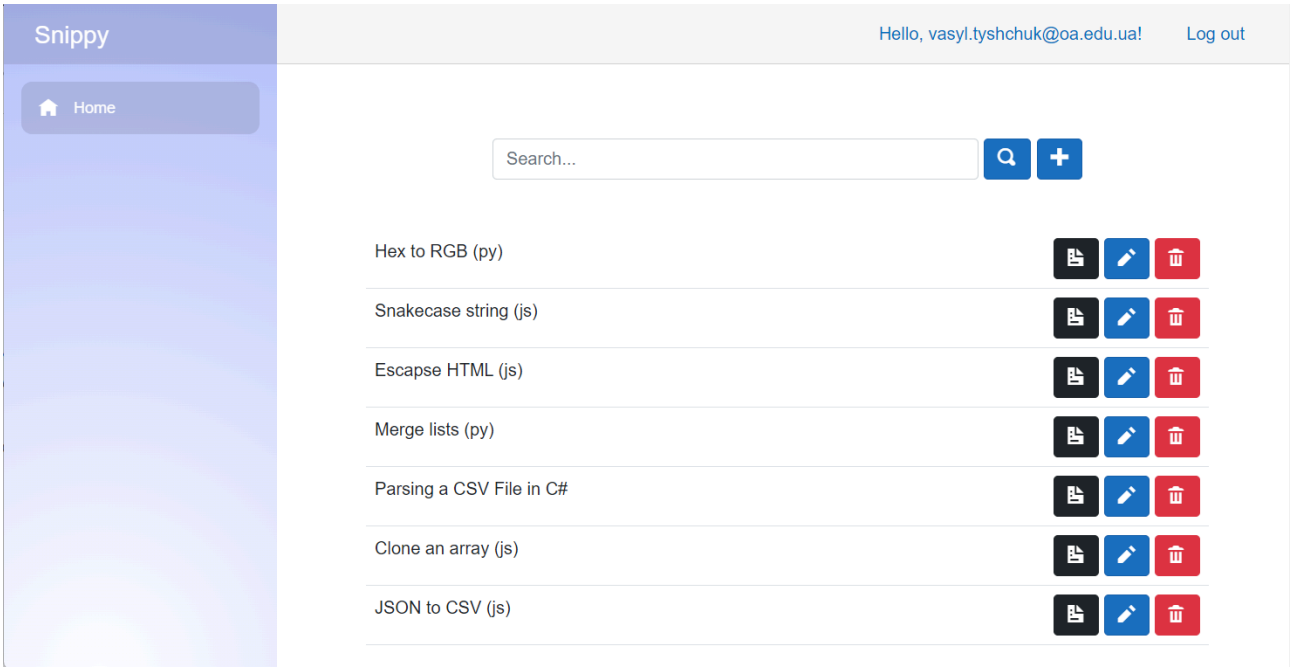

Рис. 2.8. Вигляд домашньої сторінки користувача

Джерело: створено автором

Домашня сторінка вебсайту надає короткий, але змістовний та зрозумілий опис сервісу, який відзначає його функціональні переваги. Для забезпечення зручності навігації та прискорення вступу в користування, використовується кнопка "Get Started" (Приступити до роботи/користування).

Домашня сторінки користувача надає можливість пошуку, створення, видалення та редагування сніпетів. Крім цього, доступна кнопка для швидкого копіювання вмісту сніпету без необхідності робити це вручну.

Пізніше, з метою покращення загального вигляду та користувацького досвіду вебсайту, роблячи його більш привабливим для відвідувачів, було прийнято рішення мігрувати з Bootstrap на Tailwind CSS (див. розділ 2, підпункт 2.2.1). Для цього спершу була додана підтримка Tailwind CSS до Visual Studio за допомого плагіну Tailwind CSS VS2022 Editor Support [24].

Далі налаштуємо клієнтську частину на використання Tailwind. Для цього додамо шляхи до всіх файлів шаблонів у файл tailwind.config.js:

Лістинг 2.15. Конфігурація Tailwind

```
module.exports = {
      content: ['./**/*.{razor,html}'],
      darkMode: 'selector',
      theme: {
      extend: {},
      },
      plugins: [],
}
```

Додамо директиви @tailwind до основного файлу стилів Styles/app.css:

Лістинг 2.16. Основний файл стилів Tailwind

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Тепер можна почати його використовувати в самому проєкті. Замінимо фрагменти коду з Bootstrap на Tailwind використовуючи CSS класи замість вручну створених стилів. Почнемо з домашньої сторінки вебсайту, додамо мобільну адаптацію та використаємо фрагмент коду з колекції сучасних CSS-кнопок [26]:

Лістинг 2.17. Домашня сторінка вебсайту (Index.razor)

```
<div class="fixed flex h-full w-full flex-col items-center justify-center
text-center md:px-[30%]">
      <h1 class="mb-6 text-7xl font-extrabold tracking-wide">Snippy</h1>
      <p class="mb-10 p-5 text-2xl">Snippet manager for developers designed
to effortlessly create and organize your own personal snippet collection
and have quick access to it</p>
      <a href="/app">
      <button class="group relative inline-flex h-12 items-center
justify-center overflow-hidden rounded-md bg-neutral-950 px-6 font-medium
text-neutral-200"><span>Get Started</span><div class="absolute inset-0
flex h-full w-full justify-center
[transform:skew(-12deg)_translateX(-100%)] group-hover:duration-1000
group-hover:[transform:skew(-12deg)_translateX(100%)]"><div
class="bg-white/20 relative h-full w-8"></div></div></button>
      </a>
</div>
```

Також додамо фон з колекції сучасних фонових фрагментів [25] для створення стильного та привабливого вигляду веб-сторінки:

Лістинг 2.18. Фрагмент який додає фоновий градієнт до Index.razor

```
<div class="-z-10 fixed left-0 top-0 h-full w-full"><div class="-z-10
bg-[linear-gradient(to_right,#f0f0f0_1px,transparent_1px),linear-gradient(
to_bottom,#f0f0f0_1px,transparent_1px)] absolute inset-0 h-full w-full
bg-white bg-[size:6rem_4rem]"><div
class="bg-[radial-gradient(circle_800px_at_100%_200px,#d5c5ff,transparent)
] absolute bottom-0 left-0 right-0 top-0"></div></div></div>
```

Іконки візьмемо з популярної бібліотеки Font Awesome переваги якої наведено на рисунку нижче [14].

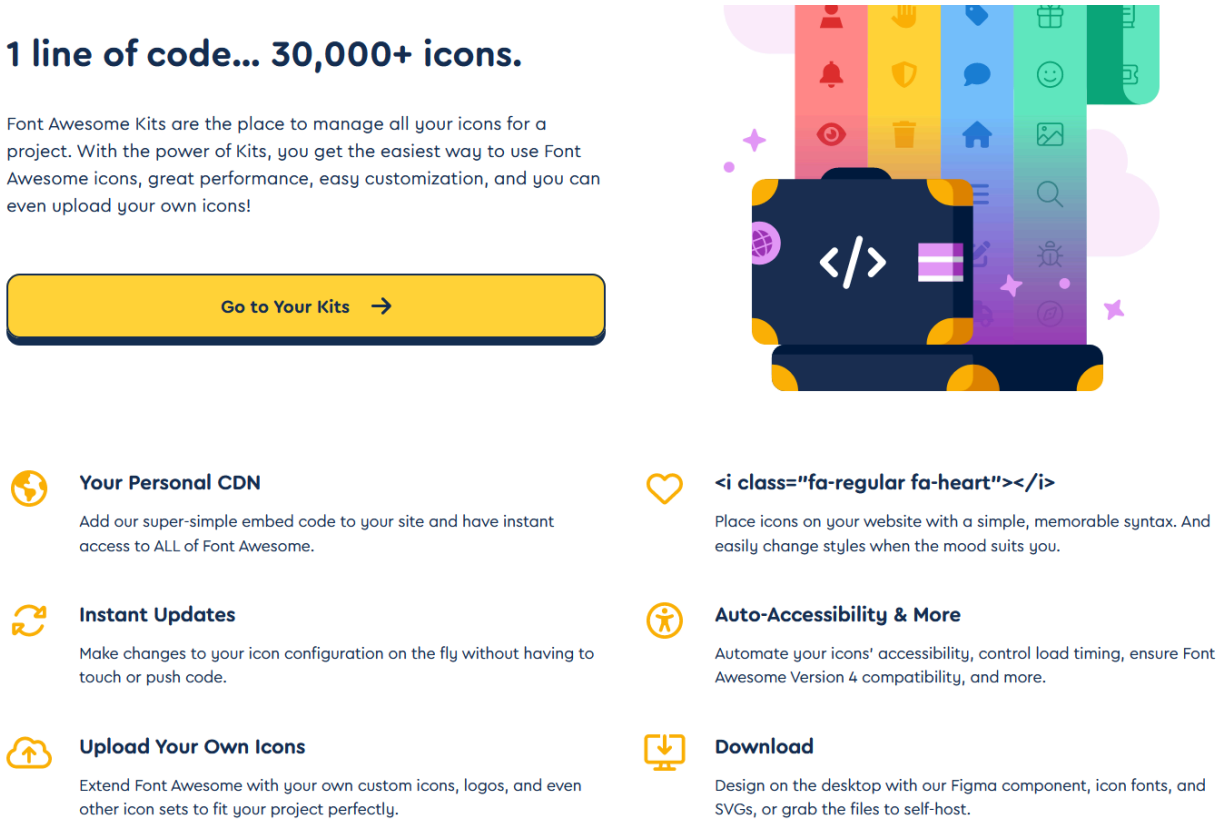

Рис. 2.9. Переваги Font Awesome [15]

Як результат отримано наступний вигляд у браузері:

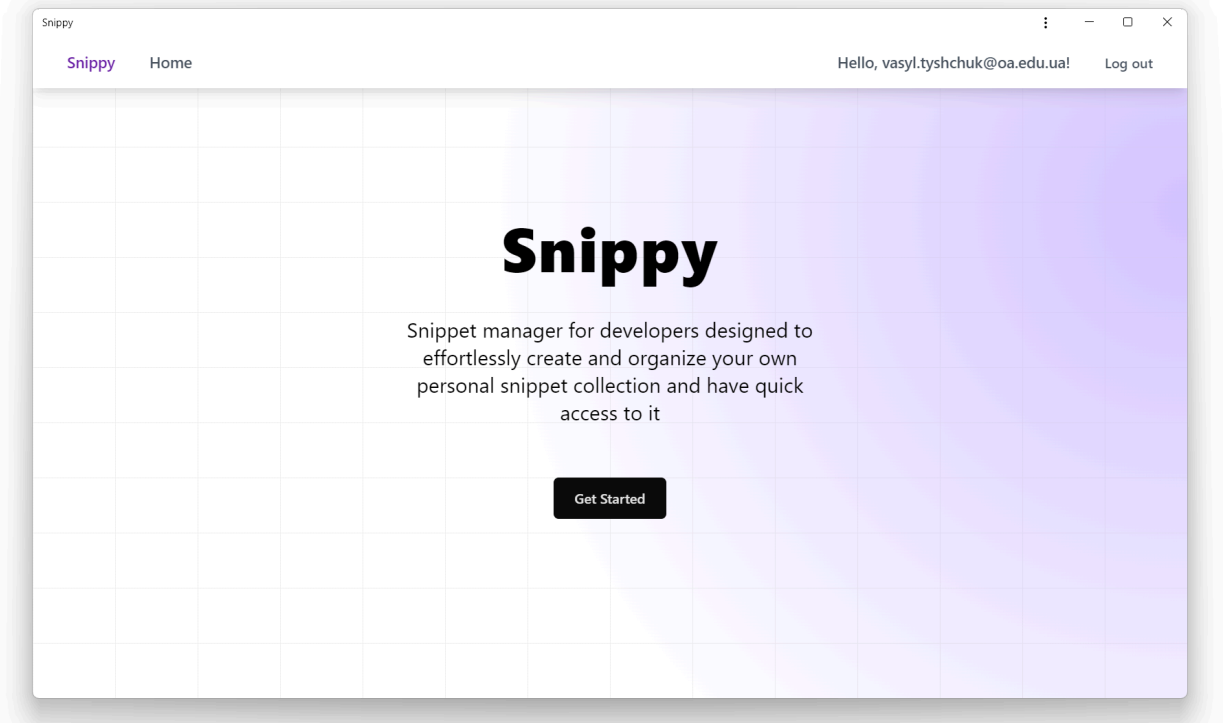

Рис. 2.10. Вигляд домашньої сторінки вебсайту

Джерело: створено автором

Покращимо домашню сторінку користувача. Додамо тіней, заокруглених кутів та змінимо колір для рядку пошуку:

Лістинг 2.19. Фрагмент коду який відповідає за рядок пошуку

```
      <div class="mb-3 flex items-center md:w-[50%]">
      <input class="w-full appearance-none rounded-lg border px-5 py-2
text-gray-700 shadow focus:outline-none focus:shadow-outline"
placeholder="Search.." @bind="searchString" @onkeyup="HandleKeyPress">
      <button class="ml-2 self-start rounded-lg bg-purple-500 p-2 shadow-md
hover:bg-purple-800" @onclick="Search">
            <i class="fa-solid fa-magnifying-glass w-6 text-white"></i>
      </button>
      <button class="ml-2 self-start rounded-lg bg-purple-500 p-2 shadow-md
hover:bg-purple-800" @onclick="CreateNewSnippet">
            <i class="fa-solid fa-plus w-6 text-white"></i>
      </button>
      </div>
```

Модифікуємо вигляд списку сніпетів використовуючи більш приємний, мінімалістичний дизайн:

Лістинг 2.20. Фрагмент коду який відповідає за список сніпетів

```
   <table class="table-auto border-collapse md:w-[50%]">
      <tbody>
            @foreach (var snippet in SnippetService.Snippets)
            {
                  <tr class="border-b border-gray-200">
                  <td class="py-2">
                        <a class="font-semibold hover:text-purple-800"
href="/app/snippet/@snippet.Id">@snippet.Title</a>
                  </td>
                  <td class="py-2 text-right">
                        <span class="ml-2">
                              <button class="rounded-lg border bg-transparent
px-4 py-2 font-semibold text-black shadow hover:bg-purple-600
hover:text-white hover:border-transparent" @onclick="(() =>
CopyToClipboard(snippet.Id, snippet.Body))">
                              <i id="@($"copy-icon-{snippet.Id}")"
class="fa-regular fa-clipboard"></i>
                              </button>
                        </span>
```

```
                    <span class="ml-2">
                        <button class="rounded-lg border bg-transparent
px-4 py-2 font-semibold text-black shadow hover:bg-purple-600
hover:text-white hover:border-transparent" @onclick="(() =>
DeleteSnippet(snippet.Id))">
                        <i class="fa-regular fa-trash-can"></i>
                        </button>
                    </span>
                </td>
                </tr>
            }
        </tbody>
    </table>
```

Створена детальна сторінка має наступний вигляд:

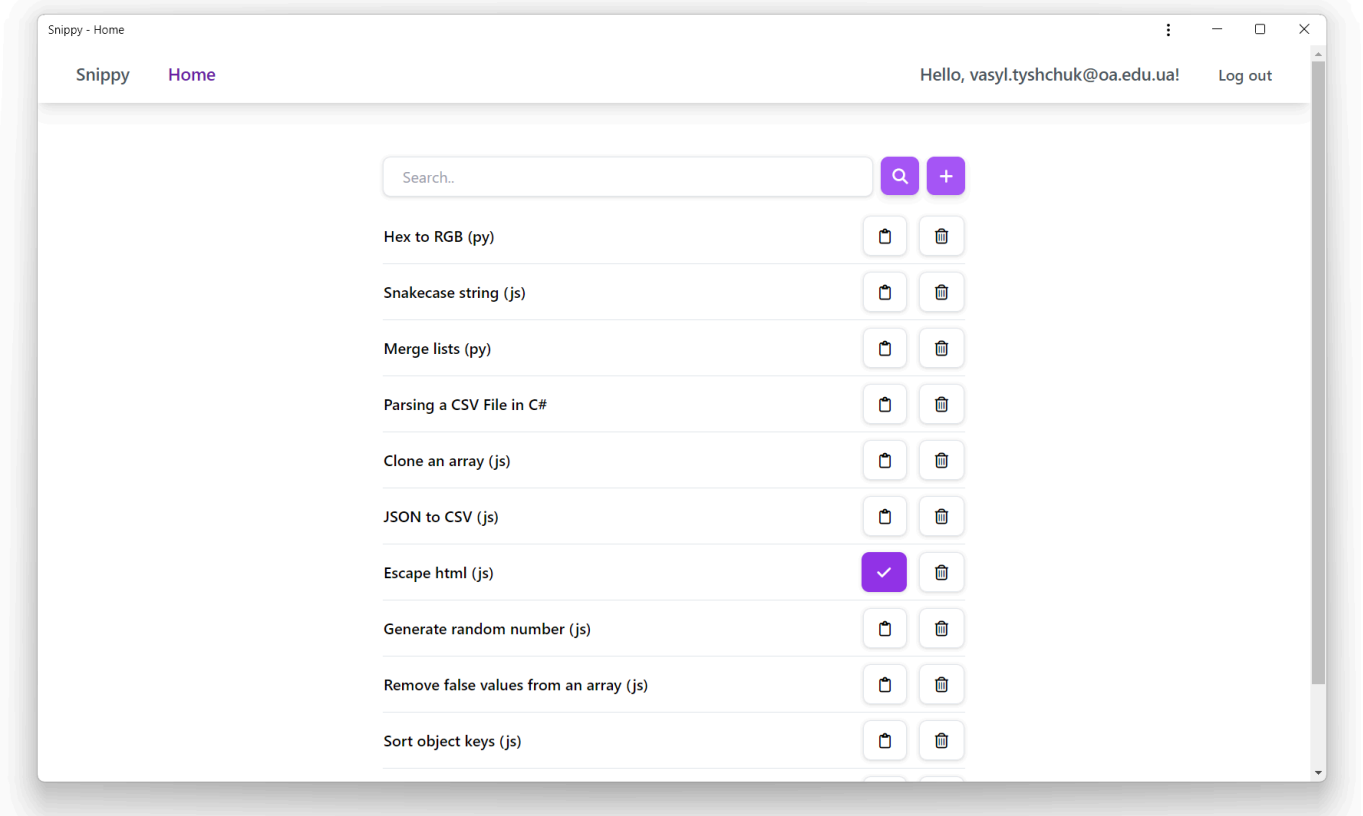

Рис. 2.11. Вигляд домашньої сторінки користувача

Джерело: створено автором

Як можна помітити стандартна бокова панель тепер відсутня оскільки вона не є потрібною в даному веб-додатку. Замість неї посилання на домашню сторінку користувача було переміщено в заголовок сайту:

Лістинг 2.21. Заголовок сайту (Shared\NavMenu.razor)

```
<header class="z-50 flex items-center justify-between bg-white px-10 py-4
shadow-lg">
      <div class="flex items-center">
      <nav class="flex text-lg font-semibold text-gray-600">
            <NavLink ActiveClass="text-purple-800" href=""
Match="NavLinkMatch.All" class="mr-10">Snippy</NavLink>
            <NavLink ActiveClass="text-purple-800"
href="/app">Home</NavLink>
      </nav>
      </div>
      <div class="flex items-center">
      <LoginDisplay />
      </div>
</header>
```

На сторінці створення та редагування сніпетів інтегруємо сторонній редактор коду з підсвічуванням синтаксису, що допомагає розробникам швидше розуміти структуру коду.

Існує декілька аналогів редакторів з підсвіткою синтаксису, кожен з яких має свої переваги та обмеження. Одним з них є редактор CodeMirror, який широко використовується в веб-додатках завдяки своїй легкості інтеграції та можливостям налаштування. Однак, в порівнянні з іншими аналогами, CodeMirror може відступати за функціональністю та швидкістю роботи.

Іншим аналогом є Ace Editor, який також має ряд переваг, таких як підтримка різних мов програмування та висока продуктивність. Однак, інтеграція Ace Editor може бути складнішою, адже він вимагає більше зусиль для налаштування та розширення.

Враховуючи ці аналоги, було обрано редактор Monaco [12] для імплементації вбудованого редактора коду з підсвіткою синтаксису. Однією з його головних переваг є зручна інтеграція з фреймворком Blazor, що робить його ідеальним вибором для ефективної розробки веб-додатків на платформі ASP.NET. Крім того, Monaco базується на технології, яка використовується в редакторі коду Visual Studio Code, що робить його інтерфейс знайомим для багатьох розробників та зручним у використанні.

В результаті порівняння аналогів та їх характеристик, обрана технологія Monaco виявилася оптимальною для вбудованого редактора коду з підсвіткою синтаксису. Вона поєднує в собі зручність інтеграції з фреймворками та знайомий інтерфейс, що робить процес редагування сніпетів коду ефективним та приємним для розробників.

Для інтеграції редактора Monaco спочатку був встановлений NuGet пакет BlazorMonaco [27] та додані відповідні скрипти до index.html відповідно до документації. Після цього був доданий компонент StandaloneCodeEditor до сторінки сніпету – Snippet.razor (див. додаток Й), налаштований на отримання інформації з бази даних та стилізований відповідно до дизайну веб-сайту. В результаті був успішно інтегрований редактор коду з підсвіткою синтаксису представлений на рисунку нижче:

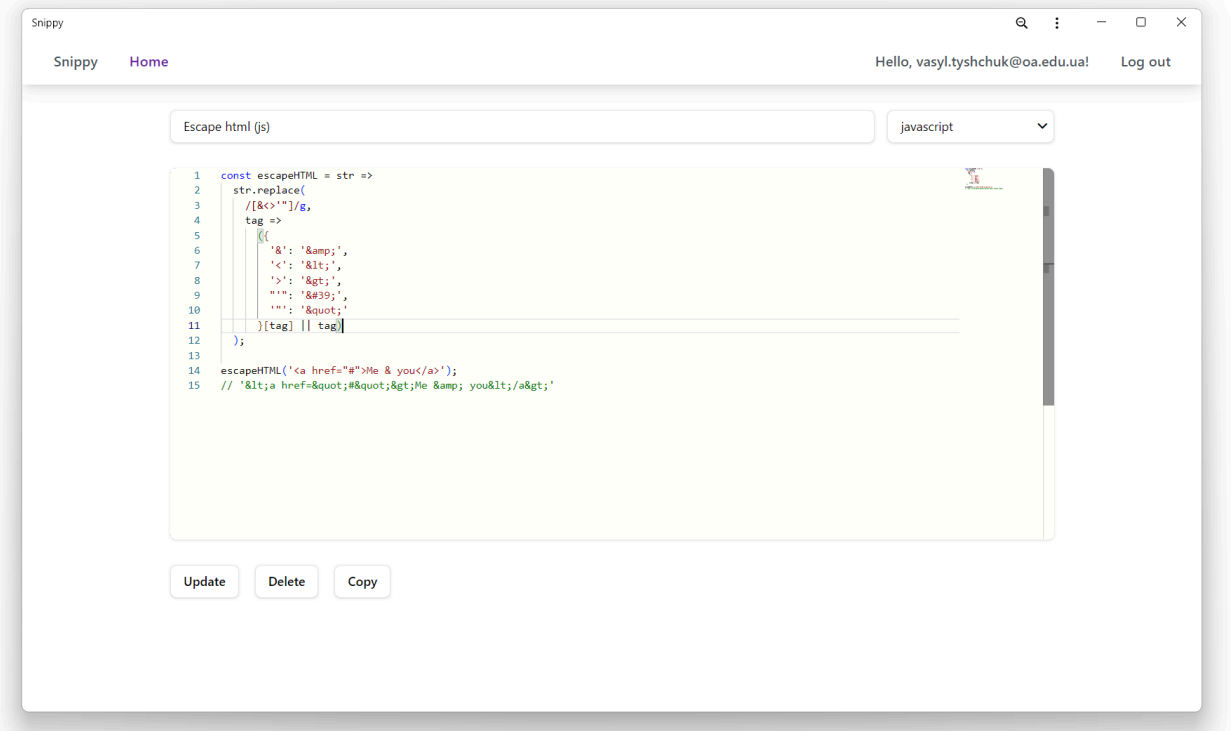

Рис. 2.12. Вигляд сторінки редагування сніпету

Джерело: створено автором

Реалізуємо власну сторінку логіну та реєстрації. Для цього натиснемо правою кнопкою миші в вікні Solution Explorer по SnippetManager.Server, після цього Add > New Scaffolded Item. В вікні вибору оберемо Identity та натиснемо Add:

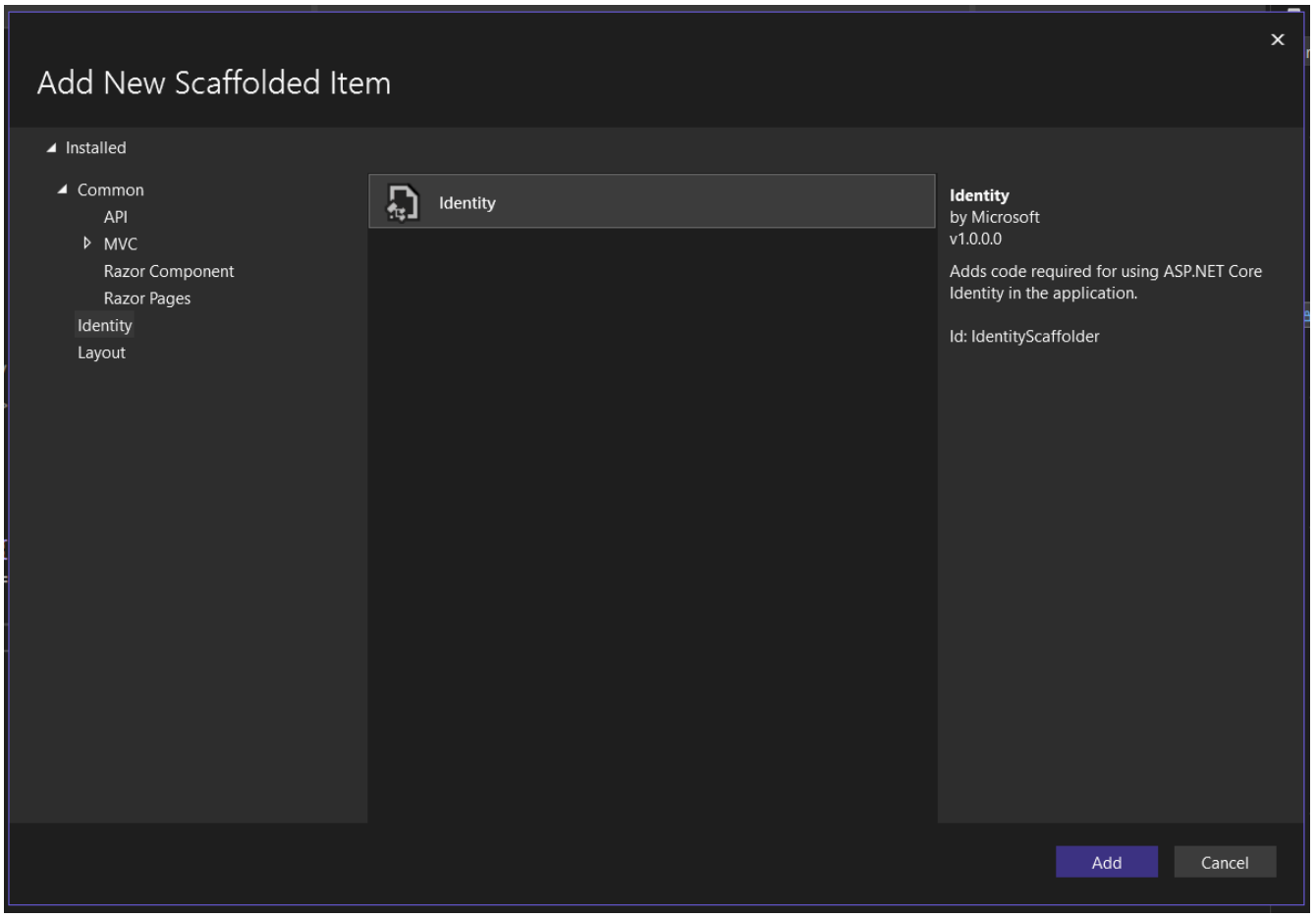

Рис. 2.13. Вікно New Scaffolded Item

Джерело: створено автором

Після цього в зявившомуся вікні Add Identity оберемо створення сторінки макета для Account\Login та Account\Register. Натиснемо Add для підтвердження операції:

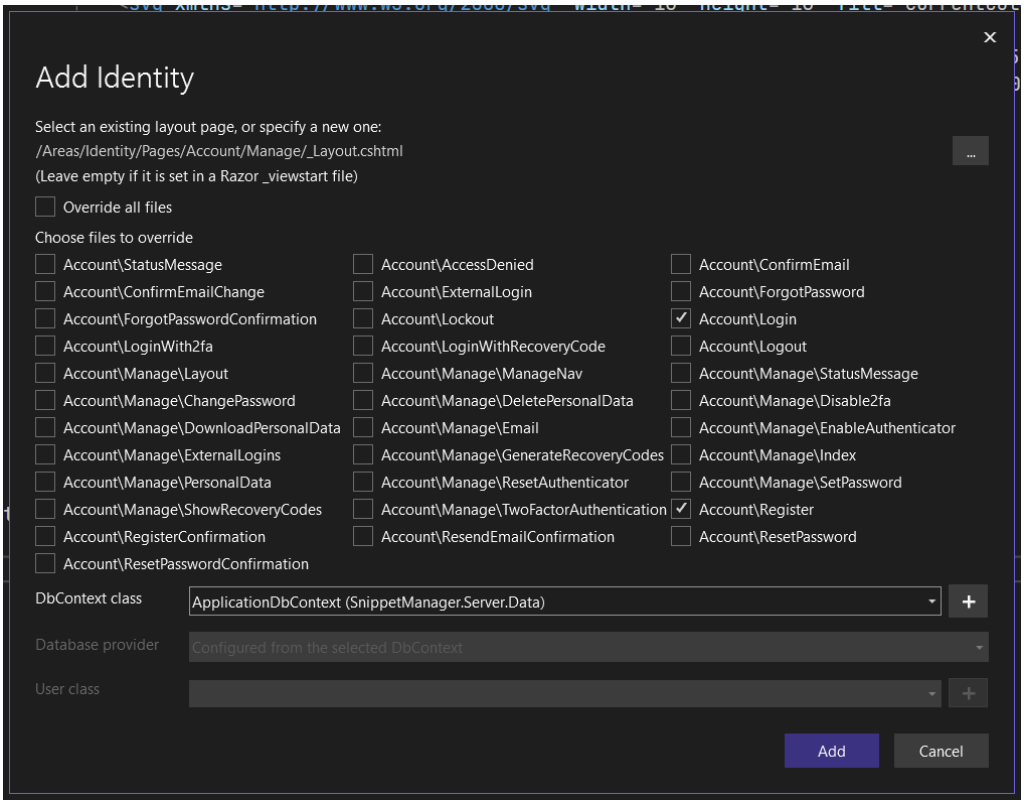

Рис. 2.14. Вікно Add Identity

Джерело: створено автором

Модифікуємо створений макет сторінки логіну та реєстрації повний код яких можна переглянути в додатку Ж та К. Як результат, отримаємо наступний вигляд сторінки логіну та реєстрації:

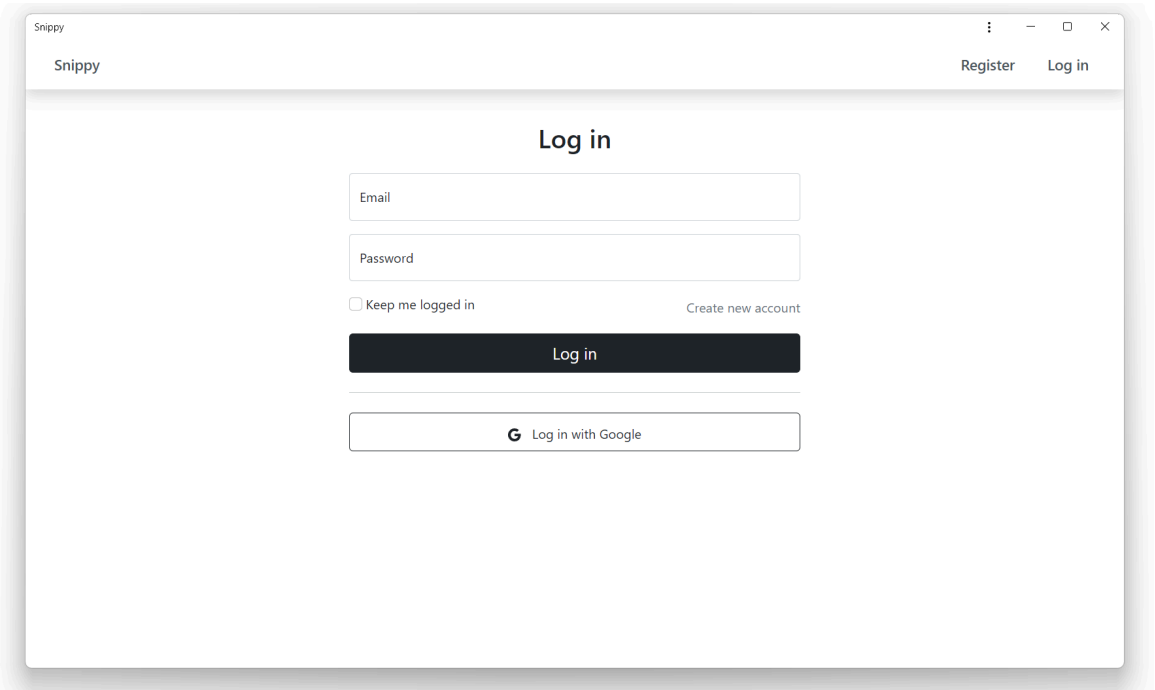

Рис. 2.15. Вікно сторінки логіну

Джерело: створено автором

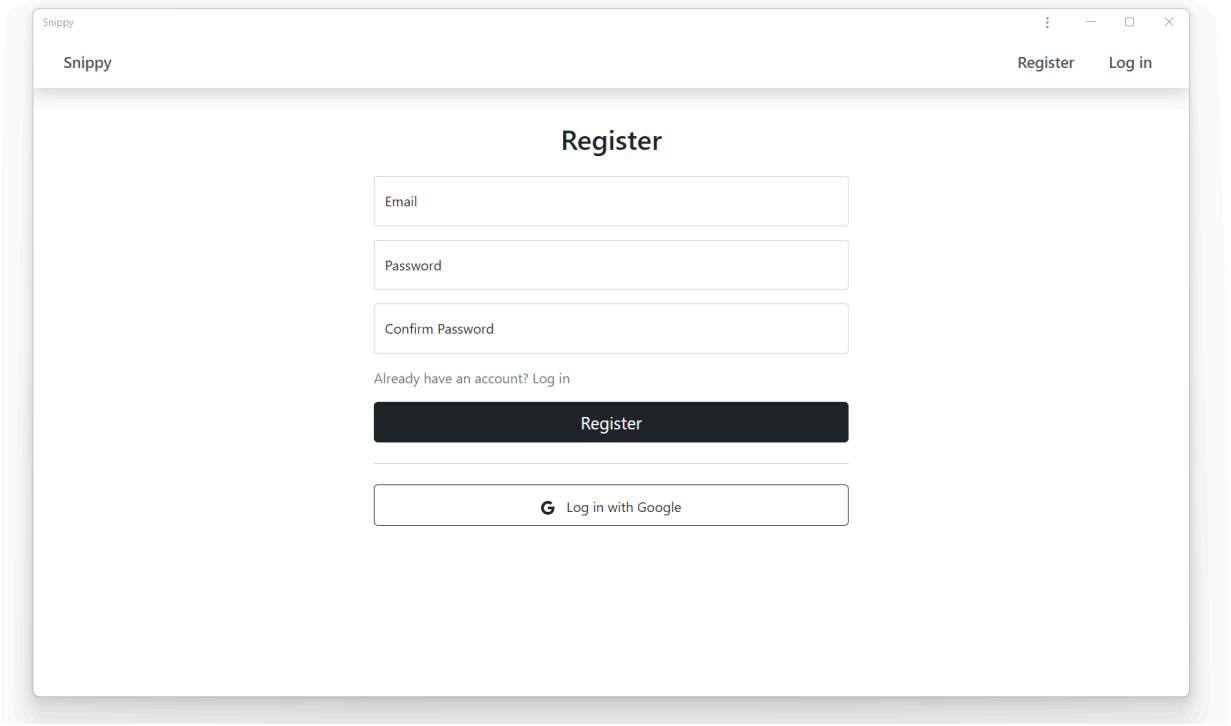

Рис. 2.16. Вікно сторінки реєстрації

Джерело: створено автором

В результаті оновлень внесених у дизайн вебсайту, було досягнуто загального поліпшення візуального сприйняття вебсайту.

2.3. DevOps частина веб-додатку “Snippy”

Обраною хмарною платформою для розгортання проєкту стала платформа Azure [13], і це рішення обрано з ряду вагомих причин.

Azure, яка являє собою хмарну платформу від Microsoft, визначається не лише своєю широкою функціональністю, але й глибокою інтеграцією з іншими продуктами та сервісами компанії. Це забезпечує високий рівень ефективності та оптимальну сумісність для проєктів, створених на Blazor.

Аналізуючи аналогічні хмарні платформи, такі як Amazon Web Services (AWS) та Google Cloud Platform (GCP), важливо враховувати конкретні потреби та характеристики проєкту. AWS визначається своєю гнучкістю та широким

функціоналом, тоді як GCP славиться своєю продуктивністю та інноваційністю. Однак, у випадку ASP.NET та враховуючи оптимальність інтеграції з Azure, вибір Azure стає логічним та стратегічним.

Azure забезпечує високий рівень масштабованості та продуктивності завдяки використанню власних центрів обробки даних та глобальної мережі серверів. Це забезпечує низьку затримку та стабільність роботи додатку для користувачів з різних регіонів світу.

Однією з ключових переваг Azure є його інтеграція з іншими продуктами Microsoft, такими як Azure DevOps для автоматизації процесів CI/CD та Azure Active Directory для керування ідентифікацією та доступом. Це спрощує процес розгортання та управління додатком.

Також Azure надає розширений спектр сервісів для контейнеризації, масштабування, безпеки та моніторингу, що важливо для високоефективного розгортання та управління веб-додатками.

Отже, обрана хмарна платформа Azure виступає оптимальним вибором для розгортання нашого проєкту, оскільки забезпечує високий рівень ефективності, інтеграції та надійності. Ця платформа надає зручний інтерфейс та деталізовані інструменти для ефективного керування та оптимізації ресурсів, роблячи її ідеальним рішенням для нашого проєкту.

Розгортання проєкту в Azure передбачає кілька етапів, які включають створення веб-додатку (Web App), бази даних та налаштування оточення. Опишемо кожен з цих етапів:

1. Створення веб-додатку (Web App):
    - У консолі Azure Portal переходимо до розділу "Створення ресурсів".
    - Вибираємо "Веб-додаток" (Web App).
    - Заповнюємо необхідні поля, такі як назва, регіон, ресурсну групу, операційна система, план служби тощо.
    - Активуємо Github Actions, обираємо потрібний акаунт, репозиторій та гілку.
    - Після створення ресурсу отримуємо URL для доступу до веб-додатку.

2. Створення бази даних:
    - У розділі "Створення ресурсів" вибираємо "База даних".
    - Заповнюємо необхідні поля, такі як назва, регіон, ресурсну групу тощо.
    - Після створення бази даних отримуємо рядок підключення для подальшого використання у веб-додатку.
3. Налаштування оточення:
    - На сторінці створеного веб-додатку переходимо до вкладки конфігурації в розділі “Налаштування”.
    - В розділі "Налаштування програми" додаємо змінні оточення, такі як ідентифікатор та секрет клієнта Google (ClientId та ClientSecret).
    - В розділі "Рядки підключення” додаємо рядок підключення до створеної бази даних.

Під час розгортання веб-додатку був використаний сервіс Github Actions який дозволяє автоматизувати процес оновлення та розгортання вебсайту за допомогою CI/CD (Неперервна Інтеграція/Неперервна Доставка). Для цього був створений файл з конфігурацією, який містить опис робочих процесів:

Лістинг 2.22. Код файлу .github/workflows/main_snippy.yml

```
name: Build and deploy ASP.Net Core app to Azure Web App - snippy
on:
  push:
      branches:
      - main
  workflow_dispatch:
jobs:
  build:
      runs-on: ubuntu-latest
      steps:
      - uses: actions/checkout@v4
      - name: Set up .NET Core
      uses: actions/setup-dotnet@v1
      with:
            dotnet-version: '7.x'
            include-prerelease: true
      - name: Build with dotnet
```

```
      run: dotnet build --configuration Release
      - name: dotnet publish
      run: dotnet publish -c Release -o ${{env.DOTNET_ROOT}}/myapp
      - name: Upload artifact for deployment job
      uses: actions/upload-artifact@v3
      with:
            name: .net-app
            path: ${{env.DOTNET_ROOT}}/myapp
  deploy:
      runs-on: ubuntu-latest
      needs: build
      environment:
      name: 'Production'
      url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
      permissions:
      id-token: write #This is required for requesting the JWT
      steps:
      - name: Download artifact from build job
      uses: actions/download-artifact@v3
      with:
            name: .net-app
      - name: Login to Azure
      uses: azure/login@v1
      with:
            client-id: ${{
secrets.AZUREAPPSERVICE_CLIENTID_8A2030E19929408E973721DA01F8AB91 }}
            tenant-id: ${{
secrets.AZUREAPPSERVICE_TENANTID_33543799F8B840A1A13084668923C967 }}
            subscription-id: ${{
secrets.AZUREAPPSERVICE_SUBSCRIPTIONID_F47A3E4C9B15489EA02F1FD25B3F1C99 }}
      - name: Deploy to Azure Web App
      id: deploy-to-webapp
      uses: azure/webapps-deploy@v2
      with:
            app-name: 'snippy'
            slot-name: 'Production'
            package: .
```

Завдяки цьому в відповідь на кожну подію push (додавання нових модифікацій до проєкту в гілку main) буде здійснюватись автоматизований процес збірки та розгортання додатку на Azure використовуючи Linux дистрибутив Ubuntu та консольний інструмент для роботи з .Net додатками - .Net CLI [14].

Після завершення цих кроків проєкт було повністю розгорнуто в Azure, готовий до використання. Ресурси можна буде моніторити та керувати ними через консоль Azure Portal, що забезпечить зручне управління та оптимізацію веб-додатку.

Переглянути онлайн версію додатка можна за посиланням https://snippy.azurewebsites.net.

Висновки до розділу 2

У цьому розділі було описано технологічний стек проєкту, тобто здійснено перелік основних технологій, які були використанні у проєкті, а саме ASP.NET, Web API, Entity Framework, Microsoft SQL Server, Blazor, Razor Pages та Tailwind CSS.

Також був описаний процес розробки клієнтської, серверної частини, та процес розгортання додатку в хмарі.

ВИСНОВКИ

Метою написання даної кваліфікаційної роботи було проєктування та розробка інтерактивного менеджера сніпетів програмних кодів, який надасть користувачам можливість ефективного управління, зберігання та швидкого доступу до часто використовуваних фрагментів коду під час розробки програмного забезпечення. Відповідно до цієї мети було здійснено кілька важливих етапів роботи, кожен з яких зробив значний внесок у досягнення загального результату.

Під час виконання кваліфікаційної роботи був здійснений аналіз наявних аналогів, що дозволило визначити основні вимоги та сформулювати завдання, необхідні для реалізації проєкту. На основі цього аналізу були розроблена серверна та клієнтська частини вебдодатку Snippy.

Серверна частина реалізована на платформі .NET з використанням таких сучасних технологій, як ASP.NET Core, Web API, EntityFramework Core та Microsoft SQL Server. Клієнтська частина була розроблена за допомогою використання фреймворку Blazor та Tailwind CSS.

Робота була організована за допомогою використання системи контролю версій Git та веб-платформи GitHub, що допомагає ефективно керувати розробкою проєкту. Для написання проєкту використовувалося програмне забезпечення Visual Studio та SQL Server Management Studio для управління базою даних, які ідеально підходять для розробки додатків на платформі .NET.

Для розробки серверної частини спершу необхідно спроектувати базу даних та імплементувати її використовуючи підхід code first за допомогою потужної ORM Entity Framework. Далі налаштувати автентифікацію та авторизацію з використання JWT токенів. Після цього створити контролери та сервіси, які забезпечують основний функціонал додатку. Контролери відповідають за обробку HTTP-запитів з клієнту, а сервіси за доступ до даних та основну бізнес-логіку.

Для розробки клієнтської частин необхідно спроектувати інтерфейс користувача та імплементувати його використовуючи фреймворк Blazor. Для стилізації інтерфейсу використати Tailwind CSS який надасть веб-сайту

привабливого та сучасного вигляду. Після цього інтегрувати редактор коду Monaco який надасть можливість зручного редагування тексту з підсвіткою синтаксису коду.

Завершальним етапом роботи було розгортання проєкту в хмарі. Для цього була обрана платформа Azure яка має широкий функціонал та хорошу сумісність з іншими продуктами Microsoft, такими як .NET. Під час розгортання проєкту на Azure важливим кроком є активація сервісу Github Actions, який дозволяє автоматизувати процес оновлення та розгортання вебсайту за допомогою CI/CD (Неперервна Інтеграція/Неперервна Доставка).

Отже, як результат виконання кваліфікаційної роботи було спроектовано та реалізовано клієнтську та серверну частину менеджера сніпетів програмних кодів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація для фреймворку .Net. URL: https://dotnet.microsoft.com/en-us (дата звернення: 20.10.2023 р.).
2. Документація для фреймворку ASP.NET. URL: https://dotnet.microsoft.com/en-us/apps/aspnet (дата звернення: 20.10.2023 р.).
3. Документація для фреймворку Blazor. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor (дата звернення: 25.10.2023 р.).
4. Документація для Web API. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/apis (дата звернення: 25.10.2023 р.).
5. “What is Swagger?”. URL: https://qagroup.com.ua/publications/what-is-swagger (дата звернення: 15.11.2023 р.).
6. Документація для Razor Pages. URL: https://learn.microsoft.com/en-us/aspnet/core/razor-pages (дата звернення: 18.11.2023 р.).
7. Документація для Entity Framework. URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 10.11.2023 р.).
8. AutoMapper документація. URL: https://docs.automapper.org/en/stable/ (дата звернення: 28.11.2023 р.).
9. OAuth. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/social (дата звернення: 02.03.2024 р.).
10. JWT. URL: https://jwt.io/ (дата звернення: 28.11.2023 р.).
11. Документація для Azure. URL: https://learn.microsoft.com/en-us/azure (дата звернення: 22.03.2024 р.).
12. Офіційний сайт редактора Monaco. URL: https://microsoft.github.io/monaco-editor (дата звернення: 28.03.2024 р.).
13. Офіційний сайт Tailwind CSS. URL: https://tailwindcss.com/ (дата звернення: 01.05.2024 р.).

14. Документація для .Net CLI. URL: https://learn.microsoft.com/en-us/dotnet/core/tools (дата звернення: 22.03.2024р.).
15. Офіційний сайт Font Awesome. URL: https://fontawesome.com (дата звернення: 10.05.2024 р.).
16. Посилання на репозиторій, розміщений у Github. URL: https://github.com/basilioss/snippet-manager (дата звернення: 20.10.2023 р.).
17. Visual Studio - продукт Microsoft. URL: https://visualstudio.microsoft.com (дата звернення: 20.10.2023 р.).
18. SQL Server Management Studio - продукт Microsoft. URL: https://shorturl.at/aeYY2 (дата звернення: 20.10.2023 р.).
19. Git - Система контролю версій. URL: https://git-scm.com (дата звернення: 28.10.2023 р.).
20. Language Integrated Query (LINQ). URL: https://shorturl.at/FQvpL (дата звернення: 28.03.2024 р.).
21. MS SQL Server. URL: https://shorturl.at/vi7R4 (дата звернення: 22.10.2023 р.).
22. How to automatically trigger GitHub Actions workflows. URL: https://shorturl.at/kIUOT (дата звернення: 28.03.2024 р.).
23. Understanding GitHub Actions. URL: https://shorturl.at/qQx8V (дата звернення: 28.03.2024 р.).
24. Tailwind CSS VS2022 Editor Support плагін. URL: https://shorturl.at/tqV5j (дата звернення: 01.05.2024 р.).
25. Колекція сучасних фонових фрагментів Tailwind. URL: https://bg.ibelick.com (дата звернення: 01.05.2024 р.).
26. Колекція сучасних фрагментів CSS-кнопок Tailwind. URL: https://buttons.ibelick.com (дата звернення: 01.05.2024 р.).
27. Редактор Monaco для Blazor. URL: https://github.com/serdarciplak/BlazorMonaco (дата звернення: 28.03.2024 р.).
28. ASP.NET Core Blazor Progressive Web Application (PWA). URL: https://shorturl.at/HI5R5 (дата звернення: 20.10.2023 р.).

ДОДАТКИ

ДОДАТОК А

Лістинг фронтенд сервісу SnippetService

```
namespace SnippetManager.Client.Services
{
    public class SnippetService : ISnippetService
    {
       private readonly HttpClient _http;
       private readonly NavigationManager _navigationManger;
       public SnippetService(HttpClient http, NavigationManager
navigationManger)
       {
            _http = http;
            _navigationManger = navigationManger;

       }
       public List<Snippet> Snippets { get ; set; } = new List<Snippet>();
       public async Task GetSnippets()
       {
            var result = await
_http.GetFromJsonAsync<List<Snippet>>("api/snippet");
            if (result is not null)
                 Snippets = result;
       }
       public async Task<Snippet?> GetSnippetById(int id)
       {
            var result = await _http.GetAsync($"api/snippet/{id}");
            if (result.StatusCode == HttpStatusCode.OK)
            {
                 return await result.Content.ReadFromJsonAsync<Snippet>();
            }
            return null;
       }
       public async Task SearchSnippets(string searchString)
       {
            var result = await
_http.GetFromJsonAsync<List<Snippet>>($"api/snippet/search?searchString={se
archString}");
            if (result is not null)
                 Snippets = result;
       }
```

```
        public async Task CreateSnippet(Snippet snippet)
        {
            await _http.PostAsJsonAsync("api/snippet", snippet);
            _navigationManger.NavigateTo("/app");
        }

        public async Task DeleteSnippet(int id)
        {
            await _http.DeleteAsync($"api/snippet/{id}");
            _navigationManger.NavigateTo("/app");
        }

        public async Task UpdateSnippet(int id, Snippet snippet)
        {
            await _http.PutAsJsonAsync($"api/snippet/{id}", snippet);
            _navigationManger.NavigateTo("/app");
        }
    }
}
```

ДОДАТОК Б

Лістинг фронтенд сервісу LanguageService

```
namespace SnippetManager.Client.Services
{
    public class LanguageService : ILanguageService
    {
       private readonly HttpClient _http;
       public LanguageService(HttpClient http)
       {
            _http = http;
       }
       public List<Language> Languages { get; set; } = new List<Language>();

       public async Task GetLanguages()
       {
            var result = await
_http.GetFromJsonAsync<List<Language>>("api/language");
            if (result is not null)
                 Languages = result;
       }
    }
}
```

ДОДАТОК В

Лістинг бекенд сервісу SnippetService

```
namespace SnippetManager.Server.Services
{
    public class SnippetService : ISnippetService
    {
       private readonly ApplicationDbContext _context;
       private readonly IMapper _mapper;

       public SnippetService(ApplicationDbContext context, IMapper mapper)
       {
            _context = context;
            _mapper = mapper;
       }
       public async Task<List<Snippet>> GetSnippets(string userId)
       {
            return await _context.Snippets
                 .Where(snippet => snippet.ApplicationUserId == userId)
                 .ToListAsync();
       }

       public async Task<Snippet?> GetSnippetById(int snippetId)
       {
            var dbSnippet = await _context.Snippets.FindAsync(snippetId);
            return dbSnippet;
       }
       public async Task<List<Snippet>> SearchSnippets(string userId,
string? searchString)
       {
            if (string.IsNullOrWhiteSpace(searchString))
            {
                 return await GetSnippets(userId);
            }

            return await _context.Snippets
                 .Where(snippet => snippet.ApplicationUserId == userId)
                 .Where(snippet =>
snippet.Title.ToLower().Contains(searchString.ToLower()))
                 .ToListAsync();
       }

       public async Task<Snippet> CreateSnippet(string userId, SnippetDto
snippetDto)
```

```
        {
            var snippet = _mapper.Map<SnippetDto, Snippet>(snippetDto);
            snippet.ApplicationUserId = userId;

            _context.Add(snippet);
            await _context.SaveChangesAsync();

            return snippet;
        }

        public async Task<Snippet?> UpdateSnippet(int snippetId, SnippetDto
snippetDto)
        {
            var dbSnippet = await _context.Snippets.FindAsync(snippetId);
            if (dbSnippet != null)
            {
                _mapper.Map(snippetDto, dbSnippet);

                await _context.SaveChangesAsync();
            }

            return dbSnippet;
        }

        public async Task<bool> DeleteSnippet(int snippetId)
        {
            var dbSnippet = await _context.Snippets.FindAsync(snippetId);
            if (dbSnippet == null)
            {
                return false;
            }

            _context.Remove(dbSnippet);
            await _context.SaveChangesAsync();

            return true;
        }
    }
}
```

ДОДАТОК Г

Лістинг бекенд сервісу LanguageService

```
namespace SnippetManager.Server.Services
{
	public class LanguageService : ILanguageService
	{
	private readonly ApplicationDbContext _context;
	private readonly IMapper _mapper;

	public LanguageService(ApplicationDbContext context, IMapper mapper)
	{
		_context = context;
		_mapper = mapper;
	}

	public async Task<List<LanguageDto>> GetLanguages()
	{
		var languages = await _context.Languages.ToListAsync();
		return _mapper.Map<List<LanguageDto>>(languages);
	}
	}
}
```

ДОДАТОК Д

Лістинг домашньої сторінки користувача

```
@page "/app"
@using Microsoft.AspNetCore.Authorization;
@inject IJSRuntime JSRuntime
@inject ISnippetService SnippetService
@inject NavigationManager NavigationManager
@attribute [Authorize]

<PageTitle>Home</PageTitle>

<div style="padding: 50px; display: flex; flex-direction: column;
align-items: center;">
    <div style="display: flex; align-items: center; width: 70%;
margin-bottom: 30px;">
       <input type="text" placeholder="Search..." class="form-control mb-3"
@bind="searchString" @onkeyup="HandleKeyPress" />
       <button class="btn btn-primary" style="align-self: flex-start;
margin-left: 5px;" @onclick="Search">
            <i class="oi oi-magnifying-glass"></i>
       </button>
       <button class="btn btn-primary" style="align-self: flex-start;
margin-left: 5px;" @onclick="CreateNewSnippet">
            <i class="oi oi-plus"></i>
       </button>
    </div>

    <table class="table">
       <tbody>
            @foreach (var snippet in SnippetService.Snippets)
            {
                 <tr>
                      <td>@snippet.Title</td>
                      <td style="text-align: right;" class="text-right">
                           <span class="ml-2">
                                <button class="btn btn-dark"
@onclick="(() => CopyToClipboard(snippet.Id, snippet.Body))">
                                     <i id="@($"copy-icon-{snippet.Id}")"
class="oi oi-document"></i>
                                </button>
                           </span>
                           <span class="ml-2">
                                <button class="btn btn-primary"
```

```
@onclick="(() => EditSnippet(snippet.Id))">
                                        <i class="oi oi-pencil"></i>
                                    </button>
                                </span>
                                <span class="ml-2">
                                    <button class="btn btn-danger"
@onclick="(() => DeleteSnippet(snippet.Id))">
                                        <i class="oi oi-trash"></i>
                                    </button>
                                </span>
                            </td>
                        </tr>
                }
            </tbody>
        </table>
</div>

@code {
    protected override async Task OnInitializedAsync()
    {
        await SnippetService.GetSnippets();
    }

    private void HandleKeyPress(KeyboardEventArgs e)
    {
        if (e.Key == "Enter")
        {
            Search();
        }
    }

    private string searchString = "";

    async void Search()
    {
        await SnippetService.SearchSnippets(searchString);
        StateHasChanged(); // Trigger a re-render of the component
    }

    void EditSnippet(int id)
    {
        NavigationManager.NavigateTo($"/app/snippet/{id}");
    }
```

```
    async void DeleteSnippet(int id)
    {
        await SnippetService.DeleteSnippet(id);
        await SnippetService.GetSnippets();
        StateHasChanged();
    }

    void CreateNewSnippet()
    {
        NavigationManager.NavigateTo("/app/snippet");
    }

    async Task CopyToClipboard(int snippetId, string snippetBody)
    {
        await JSRuntime.InvokeVoidAsync("changeCopyIconTemporary",
snippetId);
        await JSRuntime.InvokeVoidAsync("navigator.clipboard.writeText",
snippetBody);
    }
}
```

ДОДАТОК Е

Лістинг клієнт сторінки сніпета

```
@page "/app/snippet"
@page "/app/snippet/{id:int}"
@using BlazorMonaco
@using BlazorMonaco.Editor
@using BlazorMonaco.Languages
@inject ISnippetService SnippetService
@inject ILanguageService LanguageService
@inject NavigationManager NavigationManager

<EditForm Model="snippet" OnSubmit="HandleSubmit">
    <div class="row" style="padding-bottom: 20px;">
      <div class="col-md-10">
            <InputText id="title" @bind-Value="snippet.Title"
class="form-control" />
      </div>
      <div class="col-md-2">
            <select id="language" class="form-control"
@bind="selectedLanguage" @oninput="ChangeLanguage">
                  @foreach (var language in LanguageService.Languages)
                  {
                        <option
value="@language.Name">@language.Name</option>
                  }
            </select>
      </div>
    </div>
    <div style="border: 1px solid #ced4da; border-radius: 0.2rem;">
      <StandaloneCodeEditor @ref="editor"
ConstructionOptions="EditorConstructionOptions" />
    </div>
    <div style="padding-top: 20px; ">
      <button type="submit" class="btn btn-primary">@btnText</button>
      <button type="button" class="btn btn-danger"
@onclick="DeleteSnippet">
            Delete Snippet
      </button>
    </div>
</EditForm>

@code {
    [Parameter]
```

```
    public int? Id { get; set; }

    string btnText = string.Empty;
    string selectedLanguage = string.Empty;

    SnippetManager.Shared.Snippet snippet =
       new SnippetManager.Shared.Snippet { Title = "New Snippet" };

    private StandaloneCodeEditor editor;

    protected override async Task OnInitializedAsync()
    {
       btnText = Id == null ? "Save New Snippet" : "Update Snippet";
       await LanguageService.GetLanguages();
    }

    protected override async Task OnParametersSetAsync()
    {
       if (Id is not null)
       {
             var result = await SnippetService.GetSnippetById((int)Id);
             if (result is not null)
             {
                   snippet = result;
                   selectedLanguage = LanguageService.Languages
                         .FirstOrDefault(l => l.Id ==
snippet.LanguageId)?.Name ?? "plaintext";
             }
             else
             {
                   NavigationManager.NavigateTo("/app/snippet");
             }
       }
    }

    private StandaloneEditorConstructionOptions
EditorConstructionOptions(StandaloneCodeEditor editor)
    {
       this.editor = editor;

       return new StandaloneEditorConstructionOptions
             {
                   //AutomaticLayout = true,
                   Language = selectedLanguage,
```

```
                Value = snippet.Body,
            };
    }

    async Task ChangeLanguage(ChangeEventArgs e)
    {
        selectedLanguage = e.Value.ToString();
        await BlazorMonaco.Editor.Global.SetModelLanguage(await
editor.GetModel(), selectedLanguage);
    }

    async Task HandleSubmit()
    {
        var editedValue = await editor.GetValue();
        snippet.Body = editedValue;
        snippet.LanguageId = LanguageService.Languages
              .FirstOrDefault(l => l.Name == selectedLanguage)?.Id ?? 1;

        if (Id is null)
        {
              await SnippetService.CreateSnippet(snippet);
        }
        else
        {
              await SnippetService.UpdateSnippet((int)Id, snippet);
        }
    }

    async Task DeleteSnippet()
    {
        var editedValue = await editor.GetValue();
        await SnippetService.DeleteSnippet(snippet.Id);
    }
}
```

ДОДАТОК Ж

Лістинг сторінки логіну

```
@page
@model LoginModel

<div class="container">
    <div class="row justify-content-center">
      <div class="col-12 col-lg-8">
            <div class="card-body p-4">
                  <h2 class="mb-4 text-center">Log in</h2>
                  <form id="account" method="post">
                        <div asp-validation-summary="ModelOnly"
class="text-danger" role="alert"></div>
                        <div class="form-floating mb-3">
                              <input asp-for="Input.Email"
class="form-control" autocomplete="username" aria-required="true"
placeholder="name@example.com" />
                              <label asp-for="Input.Email"
class="form-label">Email</label>
                              <span asp-validation-for="Input.Email"
class="text-danger"></span>
                        </div>
                        <div class="form-floating mb-3">
                              <input asp-for="Input.Password"
class="form-control" autocomplete="current-password" aria-required="true"
placeholder="password" />
                              <label asp-for="Input.Password"
class="form-label">Password</label>
                              <span asp-validation-for="Input.Password"
class="text-danger"></span>
                        </div>
                        <div class="d-flex justify-content-between
align-items-center mb-3">
                              <label asp-for="Input.RememberMe"
class="form-label">
                                    <input class="form-check-input
text-secondary" asp-for="Input.RememberMe" />
                                    Keep me logged in
                              </label>

                              <a asp-page="./Register"
asp-route-returnUrl="@Model.ReturnUrl" class="link-secondary
text-decoration-none">Create new account</a>
```

```
                        </div>
                        <div class="d-grid">
                              <button id="login-submit" type="submit"
class="btn btn-dark btn-lg">Log in</button>
                        </div>
                  </form>
                  <hr class="my-4">
                  <form id="external-account" asp-page="./ExternalLogin"
asp-route-returnUrl="@Model.ReturnUrl" method="post">
                        <div class="d-grid">
                              @foreach (var provider in
Model.ExternalLogins!)
                              {
                                    @if (provider.Name == "Google")
                                    {
                                          <button type="submit"
name="provider" value="@provider.Name" class="btn btn-lg btn-outline-dark">
                                                <svg
xmlns="http://www.w3.org/2000/svg" width="16" height="16"
fill="currentColor" class="bi bi-google" viewBox="0 0 16 16">
                                                      <path d="M15.545
6.558a9.42 9.42 0 0 1 .139 1.626c0 2.434-.87 4.492-2.384 5.885h.002C11.978
15.292 10.158 16 8 16A8 8 0 1 1 8 0a7.689 7.689 0 0 1 5.352 2.082l-2.284
2.284A4.347 4.347 0 0 0 8 3.166c-2.087 0-3.86 1.408-4.492 3.304a4.792 4.792
0 0 0 0 3.063h.003c.635 1.893 2.405 3.301 4.492 3.301 1.078 0 2.004-.276
2.722-.764h-.003a3.702 3.702 0 0 0 1.599-2.431H8v-3.08h7.545z" />
                                                </svg>
                                                <span class="fs-6 ms-2">Log in
with Google</span>
                                          </button>
                                    }
                              }
                        </div>
                  </form>
            </div>
      </div>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

ДОДАТОК Й

Лістинг Tailwind CSS стилів сторінки сніпета

```
<div class="container mx-auto mt-8 px-20">
    <EditForm Model="snippet" OnSubmit="HandleSubmit">
      <div class="grid-cols-10 grid gap-4 pb-8">
            <div class="col-span-8">
                  <InputText id="title" @bind-Value="snippet.Title"
class="w-full rounded-lg border px-4 py-2 shadow focus:outline-none
focus:shadow-outline" />
            </div>

            <div class="col-span-2">
                  <div class="relative">
                        <select id="language" class="block w-full
appearance-none rounded-lg border bg-white px-4 py-2 shadow
focus:outline-none focus:shadow-outline" @bind="selectedLanguage"
@oninput="ChangeLanguage">
                              @foreach (var language in
LanguageService.Languages)
                              {
                                    <option
value="@language.Name">@language.Name</option>
                              }
                        </select>
                        <div class="pointer-events-none absolute inset-y-0
right-0 flex items-center px-2">
                              <i class="fa-solid fa-angle-down"></i>
                        </div>
                  </div>
            </div>
      </div>

      <div class="mb-8 overflow-hidden rounded-lg shadow">
            <StandaloneCodeEditor @ref="editor"
ConstructionOptions="EditorConstructionOptions" />
      </div>

      <div>
            <button type="submit" class="rounded-lg border bg-white px-4
py-2 font-semibold shadow hover:bg-purple-600 hover:text-white
hover:border-transparent">@btnText</button>
            <button type="button" class="ml-4 rounded-lg border bg-white
px-4 py-2 font-semibold shadow hover:bg-purple-600 hover:text-white
```

```
hover:border-transparent" @onclick="DeleteSnippet">
                    Delete
                </button>
                <button type="button" class="ml-4 rounded-lg border bg-white
px-4 py-2 font-semibold shadow hover:bg-purple-600 hover:text-white
hover:border-transparent focus:outline-none focus:shadow-outline"
@onclick="(() => CopyToClipboard(snippet.Id, snippet.Body))">
                    Copy
                </button>
        </div>
    </EditForm>
</div>
```

ДОДАТОК К

Лістинг сторінки реєстрації

```
@page
@model RegisterModel

<div class="container">
    <div class="row justify-content-center">
      <div class="col-12 col-lg-8">
           <div class="card-body p-4">
                <h2 class="mb-4 text-center">Register</h2>
                <form id="registerForm"
asp-route-returnUrl="@Model.ReturnUrl" method="post">
                     <div asp-validation-summary="ModelOnly"
class="text-danger" role="alert"></div>
                     <div class="form-floating mb-3">
                          <input asp-for="Input.Email"
class="form-control" autocomplete="username" aria-required="true"
placeholder="name@example.com" />
                          <label asp-for="Input.Email"
class="form-label">Email</label>
                          <span asp-validation-for="Input.Email"
class="text-danger"></span>
                     </div>
                     <div class="form-floating mb-3">
                          <input asp-for="Input.Password"
class="form-control" autocomplete="new-password" aria-required="true"
placeholder="password" />
                          <label asp-for="Input.Password"
class="form-label">Password</label>
                          <span asp-validation-for="Input.Password"
class="text-danger"></span>
                     </div>
                     <div class="form-floating mb-3">
                          <input asp-for="Input.ConfirmPassword"
class="form-control" autocomplete="new-password" aria-required="true"
placeholder="password" />
                          <label asp-for="Input.ConfirmPassword">Confirm
Password</label>
                          <span
asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
                     </div>
                     <div class="d-flex justify-content-between
align-items-center mb-3">
```

```
                        <a asp-page="./Login"
asp-route-returnUrl="@Model.ReturnUrl" class="link-secondary
text-decoration-none">Already have an account? Log in</a>
                    </div>
                    <div class="d-grid">
                        <button id="registerSubmit" type="submit"
class="btn btn-dark btn-lg">Register</button>
                    </div>
                </form>
                <hr class="my-4">
                <form id="external-account" asp-page="./ExternalLogin"
asp-route-returnUrl="@Model.ReturnUrl" method="post">
                    <div class="d-grid">
                        @foreach (var provider in
Model.ExternalLogins!)
                        {
                            @if (provider.Name == "Google")
                            {
                                <button type="submit"
name="provider" value="@provider.Name" class="btn btn-lg btn-outline-dark">
                                    <svg
xmlns="http://www.w3.org/2000/svg" width="16" height="16"
fill="currentColor" class="bi bi-google" viewBox="0 0 16 16">
                                        <path d="M15.545
6.558a9.42 9.42 0 0 1 .139 1.626c0 2.434-.87 4.492-2.384 5.885h.002C11.978
15.292 10.158 16 8 16A8 8 0 1 1 8 0a7.689 7.689 0 0 1 5.352 2.082l-2.284
2.284A4.347 4.347 0 0 0 8 3.166c-2.087 0-3.86 1.408-4.492 3.304a4.792 4.792
0 0 0 0 3.063h.003c.635 1.893 2.405 3.301 4.492 3.301 1.078 0 2.004-.276
2.722-.764h-.003a3.702 3.702 0 0 0 1.599-2.431H8v-3.08h7.545z" />
                                    </svg>
                                    <span class="fs-6 ms-2">Log in
with Google</span>
                                </button>
                            }
                        }
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

ДОДАТОК Л

Лістинг сніпет контроллера

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class SnippetController : ControllerBase
{
	private readonly ISnippetService _snippetService;

	public SnippetController(ISnippetService snippetService)
	{
		_snippetService = snippetService;
	}

	[HttpGet]
	public async Task<List<Snippet>> GetSnippets()
	{
		return await _snippetService.GetSnippets(GetUserId(User));
	}

	[HttpGet("{id}")]
	public async Task<Snippet?> GetSnippetById(int id)
	{
		return await _snippetService.GetSnippetById(id);
	}

	[HttpGet("search")]
	public async Task<List<Snippet>> SearchSnippets(string? searchString)
	{
		return await _snippetService.SearchSnippets(GetUserId(User),
searchString);
	}

	[HttpPost]
	public async Task<Snippet?> CreateSnippet(SnippetDto createSnippetDto)
	{
		return await _snippetService.CreateSnippet(GetUserId(User),
createSnippetDto);
	}

	[HttpPut("{id}")]
	public async Task<Snippet?> UpdateSnippet(int id, SnippetDto snippet)
```

```
        {
            return await _snippetService.UpdateSnippet(id, snippet);
        }

        [HttpDelete("{id}")]
        public async Task<bool> DeleteSnippet(int id)
        {
            return await _snippetService.DeleteSnippet(id);
        }
        private string GetUserId(IPrincipal user)
        {
            var userIdClaim = (user.Identity as
ClaimsIdentity)?.FindFirst(ClaimTypes.NameIdentifier);
            return userIdClaim?.Value;
        }
}
```