

<https://doi.org/10.30857/2786-5371.2023.5.4>

УДК 004.75

СКІДАН В. В., НІКОНОВ О. Я., ВОЛІВАЧ А. П., ПАВЛЕНКО В. М.

Київський національний університет технологій та дизайну, Україна

ДОСЛІДЖЕННЯ ХМАРНИХ МІКРОСЕРВІСІВ НА БАЗІ ТЕХНОЛОГІЇ ASP.NET CORE

Мета. Дослідження хмарних мікросервісів на базі технології ASP.NET Core та використання архітектурного шаблону Model-View-ViewModel (MVVM), оцінка технічних переваг.

Методика. Дослідження хмарних мікросервісів на базі технології ASP.NET Core здійснюється на основі методів і алгоритмів аналізу програмних систем з метою покращення їх якості, безпеки та продуктивності.

Результати. В результаті дослідження хмарних мікросервісів на базі технології ASP.NET Core проаналізовано ефективність використання архітектурного шаблону MVVM. Архітектурний шаблон MVVM дозволяє розділити інтерфейс програми, базову презентацію та бізнес-логіку на три окремі класи: представлення, яке інкапсулює інтерфейс і логіку інтерфейсу; модель представлення, яка інкапсулює логіку презентації та стан; модель, яка інкапсулює бізнес-логіку та дані програми. Шаблон дозволяє створювати додатки, які є більш масштабованими і керованими, а також спростити процес тестування, підтримку та розвиток додатку. Досліджено найкращі практики для розробки та обслуговування мікросервісів в хмарі з використанням ASP.NET Core. На базі технології ASP.NET Core та архітектурного шаблону MVVM розроблено веб-додаток «Онлайн галерея» для роботи з фото-контентом.

Наукова новизна. Запропоновано використання архітектурного шаблону MVVM для побудови хмарних мікросервісів і технології ASP.NET Core. Досліджено переваги використання ASP.NET Core в контексті хмарних мікросервісів.

Практична значимість. Проведені дослідження дозволяють оцінити переваги при впровадженні ASP.NET Core для хмарних мікросервісів, що є важливим для архітекторів програмного забезпечення, розробників та ІТ-компаній, в цілому. Отримані результати дозволяють приймати обґрунтовані рішення при проектуванні хмарних мікросервісів на базі технології ASP.NET Core, і як наслідок, розробляти більш ефективні, масштабовані та безпечні програмні системи. Отримані результати є основою для майбутніх досліджень та ефективних реалізацій у постійно еволюціонуючому середовищі хмарних обчислень та мікросервісів.

Ключові слова. веб-додаток; веб-сервіс; хмарні мікросервіси; технологія ASP.NET Core; архітектурний шаблон MVVM.

Вступ. Підвищення продуктивності, безпеки та масштабованості програмних систем хмарних мікросервісів є дуже актуальними й практичними задачами, що пов'язано з ростом популярності хмарних обчислень та мікросервісів [1–11].

На сьогодні пропонуються ефективні підходи та методи, що дозволяють вирішувати наступні задачі [1–8]: контейнеризація й оркестрація – використання контейнерів (наприклад, Docker) дозволяє упаковувати додатки з усім оточенням в стандартизовані середовища, а оркестрація (наприклад, Kubernetes) управляє розгортанням, масштабуванням і керуванням контейнерами, що полегшує автоматизацію і розширення хмарних мікросервісів; автоматизація й Continuous Integration та Continuous Deployment (CI/CD) – використання практик CI/CD дозволяє автоматизувати процеси тестування, збирання та розгортання мікросервісів, що допомагає підвищити продуктивність і забезпечити швидке внесення змін; моніторинг і аналітика – важливий елемент для забезпечення якості та продуктивності системи, використання інструментів моніторингу (наприклад, Prometheus, Grafana) дозволяє відстежувати роботу мікросервісів в реальному часі та виявляти проблеми.

Слід зазначити, що безпосереднє використання мікросервісної архітектури – розділення додатка на невеликі мікросервіси зменшує вплив збоїв на весь додаток і дозволяє розробникам працювати над окремими компонентами паралельно [1–6].

Загалом, поєднання мікросервісної архітектури й хмарних обчислень може сприяти підвищенню швидкості розробки, масштабованості та надійності додатків, що є критичними аспектами у сучасному інформаційному середовищі.

Постановка задачі. Метою роботи є дослідження хмарних мікросервісів на базі технології ASP.NET Core та використання архітектурного шаблону MVVM, оцінка технічних переваг.

Цілі роботи:

- дослідження основних концепцій та принципів хмарних мікросервісів;
- дослідження архітектурного шаблону MVVM в порівнянні з іншими архітектурними підходами;
- дослідження концепції поєднання ASP.NET Core та архітектурного шаблону MVVM;
- дослідження можливості інтероперабельності та взаємодії між різними мікросервісами, які використовують ASP.NET Core та архітектурний шаблон MVVM.

Результати дослідження та висновки, отримані на основі визначених цілей, дозволять приймати обґрунтовані рішення щодо використання ASP.NET Core та архітектурного шаблону MVVM в конкретних сценаріях розробки хмарних мікросервісів.

Результати дослідження. В ході дослідження розглянуто трендові технології розробки веб-додатків. Встановлено, що частина технологій (платформ) мають відкритий вихідний код, такі як Java та PHP, тоді як інші, наприклад, ASP.NET Model-View-Controller (MVC), є закритими системами. Багато веб-розробників використовують саме ASP.NET MVC для створення веб-додатків, хоча останній реліз фреймворку – ASP.NET Core – пропонує значно більше переваг для веб-розробки, ніж його попередник. ASP.NET Core є крос-платформним фреймворком з відкритим вихідним кодом, створеним командою розробників у складі компанії Microsoft, а також за участю активної спільноти розробників. Це, по суті, перетворення оригінального ASP.NET, що об'єднує структуру MVC та Web API в єдиний комплексний фреймворк [7] (рис. 1). Розгортання мікросервісів та їх інтеграція з ASP.NET Core можливі завдяки хмарним технологіям та платформам таких як: Microsoft Azure, Amazon Web Services, Google Cloud Platform, Docker Swarm, Red Hat OpenShift, IBM Cloud та ін. Для інтеграції з ASP.NET Core створюються контейнери для кожного мікросервісу, а потім використовується оркестратор контейнерів (наприклад, Kubernetes, Docker Swarm або OpenShift) для їх керування та розгортання. ASP.NET Core ефективно працює з методами контейнеризації, оскільки має незалежність від конкретної інфраструктури і адаптується до різних хмарних платформ [1–8].

Особливості використання архітектурних шаблонів MVC та MVVM з хмарними мікросервісами та відмінності в порівнянні з іншими підходами до розробки додатків:

1. *Розділення функціональності.* Основна ідея застосування шаблонів MVC та MVVM полягає в розділенні функціональності додатка на окремі компоненти, такі як Model (модель), View (представлення) і Controller/ViewModel (контролер/модель представлення). При використанні хмарних мікросервісів кожен мікросервіс може використовувати власну реалізацію вищенаведених компонентів.

2. *Комунікація між мікросервісами.* Однією з ключових відмінностей використання архітектурних шаблонів з хмарними мікросервісами є питання комунікації між різними сервісами. Архітектура повинна передбачати механізми для обміну даними та подіями між

мікросервісами і які компоненти (наприклад, Controller або ViewModel) відповідають за цю комунікацію.

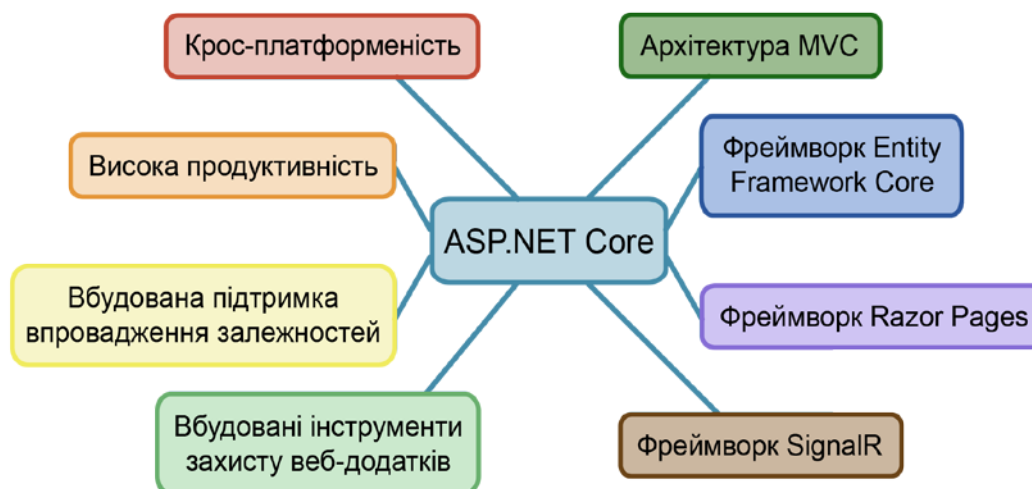


Рис. 1. Можливості технології ASP.NET Core

3. *Управління інтерфейсами.* При використанні хмарних мікросервісів один додаток може складатися з декількох інтерфейсів користувача, які можуть бути реалізовані як окремі мікросервіси або компоненти. Використання шаблонів MVC або MVVM підвищує ефективність процесу організації логіки для кожного з цих інтерфейсів.

4. *Паралельна розробка.* Розподілена структура дозволяє різним командам розробників працювати над різними мікросервісами незалежно. MVC та MVVM забезпечують чіткий розділ обов'язків між розробниками, що полегшує паралельну розробку.

Підсумовуючі вищенаведене, використання архітектурних шаблонів MVC та MVVM з хмарними мікросервісами є доцільним, але вимагає уваги до деталей взаємодії між компонентами та дотримання сучасних практик розробки мікросервісних додатків.

Слід зауважити, що важливими аспектами для забезпечення ефективності й витратоспоживання є масштабування та оптимізація мікросервісів. Сучасними підходами масштабування мікросервісів є горизонтальне масштабування, контейнеризація, оркестрація контейнерів, автоматичне масштабування. Для оптимізації мікросервісів – профілювання та оптимізація коду, кешування, моніторинг і логування, моніторинг та аналітика витрат тощо [1–7]. Загалом, оптимізація та масштабування мікросервісів – це постійний процес, який вимагає уваги до деталей та постійного моніторингу для забезпечення ефективності та зниження витрат.

Також проведено аналіз ефективності використання архітектурного шаблону MVVM для хмарних мікросервісів. Встановлено, що поєднання ASP.NET Core та архітектурного шаблону MVVM дозволяє розробникам створювати веб-додатки з ефективним розподілом задач між серверною і клієнтською частиною. При цьому інтероперабельність та взаємодія між різними мікросервісами, які використовують ASP.NET Core й архітектурний шаблон MVVM, є ключовими аспектами розробки розподілених систем. Розробники програмних систем приймають шаблон проектування MVVM (рис. 2) зі схожими мотивами об'єктно-орієнтованого програмування.

За результатами аналізу виділено ключові причини використання архітектурного шаблону MVVM для хмарних мікросервісів [1–6]:

1. *Розділення проблем.* Шаблон MVVM дозволяє розділити проблему на окремі компоненти. Model представляє дані та бізнес-логіку, View відображає інтерфейс користувача,

а ViewModel визначає взаємодію між ними, що допомагає зробити код більш організованим і зрозумілим.

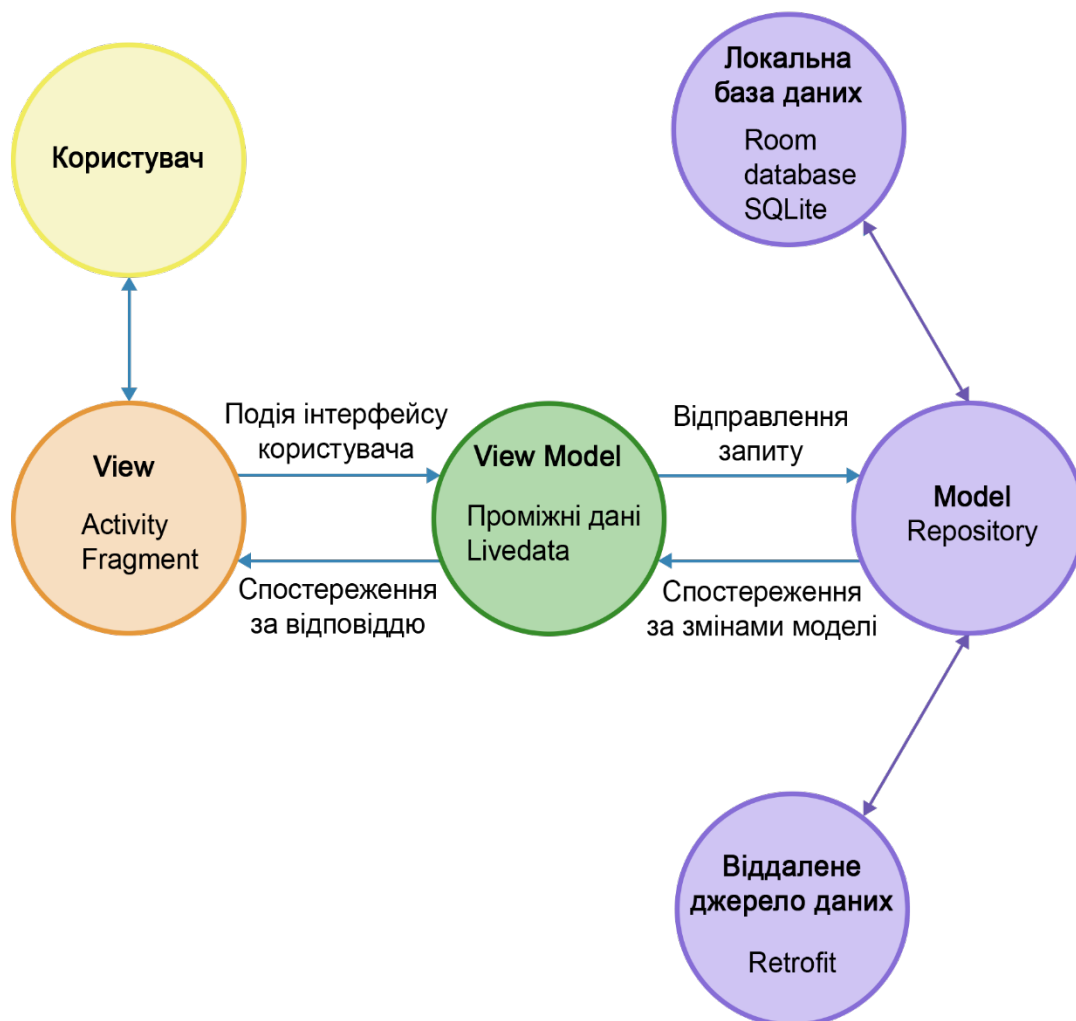


Рис. 2. Архітектурний шаблон Model-View-ViewModel

2. *Можливість повторного використання.* Шаблон MVVM сприяє створенню автономних компонентів, які можна повторно використовувати в різних частинах програми або навіть в інших проєктах, що полегшує розробку та знижує витрати часу.

3. *Зручність обслуговування.* Якщо логіка програми обгорнута в невеликі самостійні класи, то спрощується процес внесення змін і виправлення помилок, що дозволяє зосередитися на окремих компонентах без необхідності редагувати всю програму.

4. *Тестування.* Якщо логіка розділена на класи, то процес тестування стає більш ефективним. Нескладно створити набір тестів для перевірки кожної властивості та методу в окремих класах. Без використання глобальних змінних, тестування стає набагато надійнішим.

5. *Підтримка інтерфейсу та розширюваність.* MVVM робить код більш гнучким і розширюваним, що дозволяє ефективно взаємодіяти з різними інтерфейсами користувача, що особливо корисно в сучасних додатках, які можуть працювати на різних платформах та пристроях.

6. *Уніфікація команд та подій.* MVVM спрощує управління командами та подіями в програмі. Використання команд дозволяє структурувати та групувати дії, які виконуються при взаємодії з користувачем, що робить код більш організованим, зрозумілим і керованим.

7. Підтримка паралельної розробки. Шаблон MVVM розділяє функціональність додатку на компоненти, що дозволяє розробникам працювати над різними частинами програми паралельно, не заважаючи одне одному.

8. Ефективність роботи з дизайном. View та ViewModel дозволяють розділити логіку програми та її представлення, що робить співпрацю з дизайнерами більш ефективною, оскільки можна працювати над інтерфейсом без втручання у логіку програми.

Загалом, прийняття шаблону проектування MVVM базується на розподілі відповідальностей, полегшує розробку, підтримку і тестування програмного забезпечення та робить процес розробки більш ефективним й організованим.

За результатами аналізу архітектурного шаблону MVC для хмарних мікросервісів встановлено, що його концепція представляє розділення додатку на компоненти Model, View і Controller, кожен з яких виконує свою роль у додатку (рис. 3) [1–7].

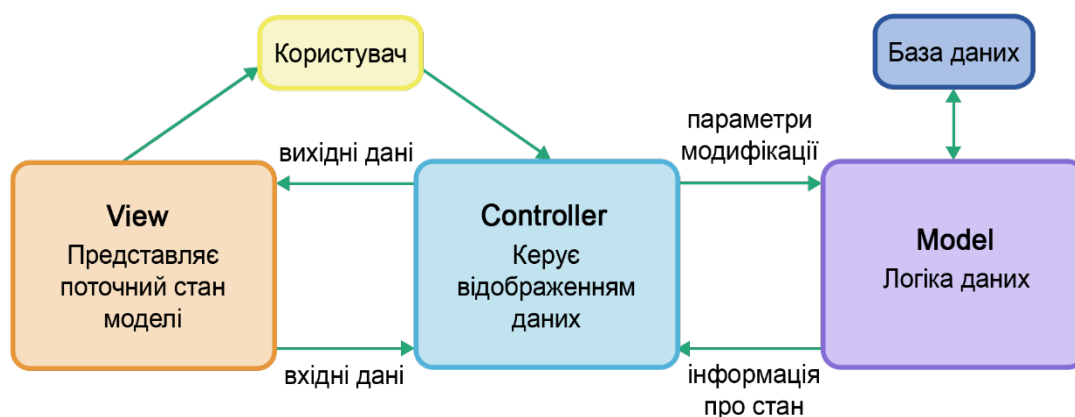


Рис. 3. Архітектурний шаблон Model-View-Controller

Model представляє собою компонент, який відповідає за управління даними та бізнес-логікою, не залежить від інтерфейсу користувача. Може включати логіку, пов'язану з доступом до бази даних, зберігання та отримання даних. При цьому бізнес-логіка інкапсульована в межах Model разом з логікою реалізації для збереження стану програми.

View відповідає за відображення інформації користувачу і відображає дані з Model та забезпечує інтерфейс для взаємодії користувача з додатком. Слід зазначити, що View не повинен містити бізнес-логіки, оскільки він відображає дані, які надаються Model. У випадку, якщо необхідно виконати велику кількість операцій з логікою у файлах перегляду для відображення даних складної моделі використовується компонента перегляду, моделі перегляду або шаблон перегляду.

Controller діє як посередник між Model та View, приймає вхідні дані від користувача через інтерфейс, виконує необхідні операції, ініціює взаємодію з моделлю та визначає, який вид має бути відображений. Controller відповідає за обробку подій та керування потоком додатка.

Узагальнюючи концепцію архітектурного шаблону MVC, а саме розділення додатку на Model, View і Controller, необхідно наголосити на практичній складовій – це полегшення розробки, тестування та підтримки коду. Архітектурний шаблон MVC дозволяє підтримувати прозору структуру, а командам розробників працювати над окремими частинами додатка паралельно.

Архітектурні шаблони MVC і MVVM використовуються для розробки програмного забезпечення з графічним інтерфейсом користувача але мають різні концепції і структури:

1. *Розподілення обов'язків.* MVC: у шаблоні MVC компонент Model представляє дані та бізнес-логіку, View відповідає за відображення даних, а Controller є посередником між Model і View, обробляючи вхідні події та оновлюючи відображення.

MVVM: у шаблоні MVVM компонент Model також представляє дані та бізнес-логіку, але у цьому випадку є ще один компонент – ViewModel, яка містить логіку для відображення даних, відокремлену від самого View. Варто зазначити, що View в MVVM відображає дані з ViewModel і реагує на дії користувача. ViewModel і View комунікують через механізми зв'язку, зазвичай за допомогою даних прив'язок (data binding).

2. *Декларативна прив'язка даних.* MVC: в MVC прив'язка даних зазвичай реалізується розробниками вручну і Controller відповідає за оновлення відображення на основі змін в Model. MVVM: у MVVM декларативна прив'язка даних дозволяє автоматично синхронізувати дані між ViewModel і View – зміни в ViewModel автоматично відображаються без прямого втручання розробника.

3. *Тестування.* MVC: тестування у шаблоні MVC може бути складнішим, оскільки бізнес-логіка і логіка відображення зазвичай змішані в Controller. MVVM: MVVM сприяє кращій роздільності обов'язків, що робить його більш придатним для тестування. ViewModel може бути протестованим окремо від View.

4. *Зручність для розробників і дизайнерів.* MVC: у шаблоні MVC дизайнери та розробники можуть практично незалежно працювати над інтерфейсом користувача, але іноді виникають проблеми зі спільною роботою через взаємозалежність Controller і View. MVVM: MVVM робить співпрацю дизайнерів і розробників більш ефективною, оскільки дизайнери можуть працювати над View, не заважаючи розробникам, які роблять ViewModel. При цьому, декларативна прив'язка даних також спрощує цей процес.

Дослідження можливості інтероперабельності та взаємодії між різними мікросервісами, які використовують ASP.NET Core та архітектурний шаблон MVVM. Взаємодія з іншими мікросервісами відбувається через ViewModel, що містить методи або команди для виконання HTTP-запитів до інших мікросервісів, обміну даними через API або використання інших засобів взаємодії, таких як WebSocket. Використовуються бібліотеки або фреймворки для мережевої взаємодії, таких як Retrofit (для Android) або Axios (для веб-додатків). Алгоритм взаємодії: 1) створюється ViewModel, в якій визначаються методи або команди, які будуть взаємодіяти з мікросервісами; 2) виконується мережевий запит до іншого мікросервісу (наприклад, за допомогою бібліотек Retrofit для Android або Axios для веб-додатків); 3) обробка відповіді (розпакування даних з формату відповіді, перетворення їх у необхідний формат даних додатка і збереження їх у властивостях ViewModel); 4) сповіщення View про зміни у даних через механізм прив'язки даних або інших методів залежно від платформи (наприклад, LiveData для Android або WPF для Windows); 5) обробка помилок.

На базі технології ASP.NET Core та архітектурного шаблону MVVM розроблено веб-додаток «Онлайн галерея» для роботи з фото-контентом. На рис. 4 і 5 представлено екран зі списком галерей і зовнішній вигляд галерей у розвороті веб-додатку «Онлайн галерея».

Переваги використання шаблону MVVM:

1. *Збереження існуючої бізнес-логіки.* ViewModel може виступати в ролі адаптера, яка ізолює розробника від внесення значних змін у код Model, що зменшує ризики порушення функціональності моделі.

2. *Модульні тести.* MVVM сприяє покращенню тестованості коду, дозволяючи створювати модульні тести для ViewModel та Model без прив'язки до View, що полегшує автоматизоване тестування та підтримку коду.

3. *Процес зміни інтерфейсу.* Завдяки розділенню логіки View від ViewModel, можливо змінювати інтерфейс частини користувача, не впливаючи на код Model та ViewModel. Такий підхід полегшує процес оновлення та вдосконалення інтерфейсу.

4. *Незалежна робота дизайнерів і розробників.* Розділення на ViewModel і Model дозволяє дизайнерам та розробникам працювати паралельно над своїми частинами без взаємних втручань. Такий підхід сприяє продуктивності та розділенню обов'язків.

5. *Розподіл обов'язків у шаблоні MVVM.* Шаблон MVVM дозволяє чітко розділити обов'язки між Model, View та ViewModel. Такий підхід полегшує управління кодом, робить його більш структурованим та зрозумілим для команди розробників.

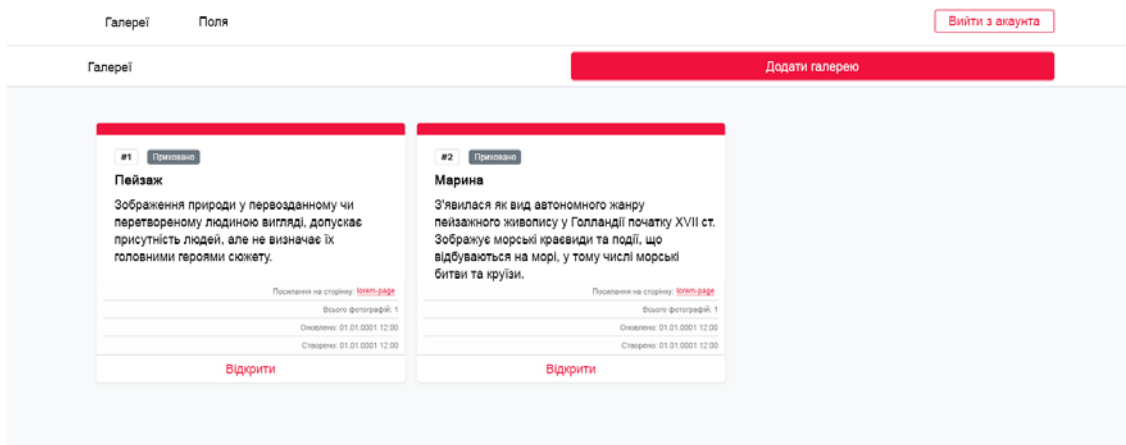


Рис. 4. Екран зі списком галерей веб-додатку «Онлайн галерея»

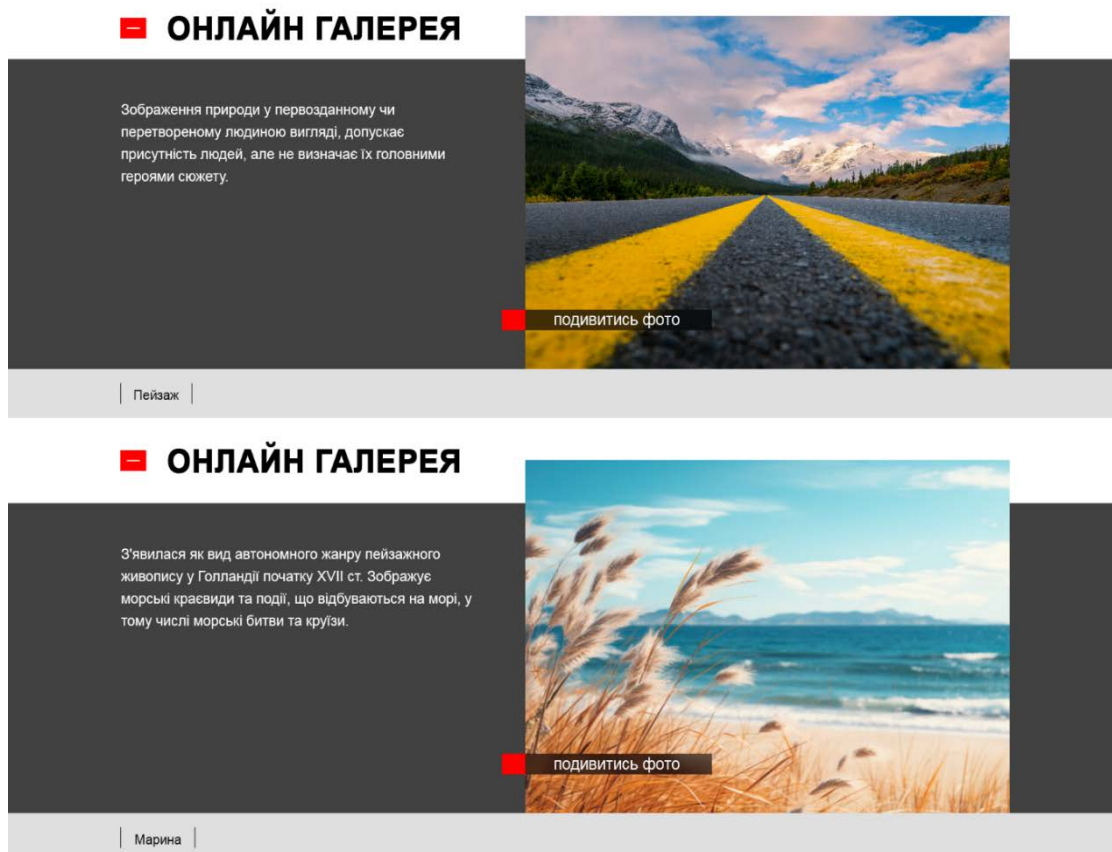


Рис. 5. Зовнішній вигляд галерей у розвороті веб-додатку «Онлайн галерея»

Таким чином, обираючи між MVC і MVVM, слід враховувати особливості проєкту та потреби користувача. MVVM може бути корисним завдяки його декларативній прив'язці даних та розділенню обов'язків, тому його слід застосовувати в більш складних інтерфейсах та проєктах. У випадку створення відносно нескладних проєктів або проєктів де декларативна прив'язка даних може бути зайвою, зручним і дієвим варіантом буде MVC.

Також досліджено переваги шаблону MVVM з ще одним популярним шаблоном MVP (Model-View-Presenter):

1. *Пряме зв'язування.* Однією з основних переваг MVVM є використання прямого зв'язування між ViewModel і View, що дозволяє автоматично оновлювати відображення елементів інтерфейсу при змінах в ViewModel та навпаки. У MVP такий механізм не є стандартним, і його потрібно реалізовувати вручну.

2. *Розділення бізнес-логіки і логіки відображення.* MVVM дозволяє чітко розділити бізнес-логіку від логіки відображення. Компонент ViewModel може бути багаторазово використано, що спрощує розробку та тестування.

3. *Зменшення шаблонного коду.* MVVM зазвичай вимагає менше шаблонного коду, оскільки не потребує явної передачі даних. Завдяки прямому зв'язуванню, оновлення інтерфейсу стає більш декларативним.

4. *Підтримка платформи.* MVVM добре підтримується на багатьох платформах і фреймворках, включаючи WPF, Xamarin, Angular та інші, що робить його універсальним і застосовним в різних проєктах.

Висновки. В результаті виконаної роботи:

1. Досліджено основні концепції та принципи хмарних мікросервісів. Показано, що поєднання мікросервісної архітектури й хмарних обчислень сприяє підвищенню швидкості розробки, масштабованості та надійності додатків, що є критичними аспектами у сучасному інформаційному середовищі.

2. Досліджено архітектурний шаблон MVVM в порівнянні з іншими архітектурними підходами. MVVM базується на розділенні відповідальностей, що полегшує розробку, підтримку і тестування програмного забезпечення.

3. Досліджено концепцію поєднання ASP.NET Core та архітектурного шаблону MVVM, що дозволяє розробникам створювати веб-додатки з ефективним розділенням задач між серверною і клієнтською частиною.

4. Досліджено можливості інтеперабельності та взаємодії між різними мікросервісами, які використовують ASP.NET Core та архітектурний шаблон MVVM. На базі технології ASP.NET Core та архітектурного шаблону MVVM розроблено веб-додаток «Онлайн галерея» для роботи з фото-контентом.

На основі конвергенції технологій ASP.NET Core і MVVM розробники програмного забезпечення можуть створювати ефективні та надійні додатки, які будуть задовольняти вимогам сучасних ринків та споживачів послуг.

References

1. Reznik, P., Dobson, J., Gienow, M. (2020). Cloud Native Transformation: Practical Patterns for Innovation. 1st Edition. O'Reilly Media. 537 p.
2. Garrison, J., Nova, K. (2017). Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment. 1st Edition. O'Reilly Media. 157 p.

Література

1. Reznik P., Dobson J., Gienow M. Cloud Native Transformation: Practical Patterns for Innovation. 1st Edition. O'Reilly Media, 2020. 537 p.
2. Garrison J., Nova K. Cloud Native Infrastructure: Patterns for Scalable Infrastructure and Applications in a Dynamic Environment. 1st Edition. O'Reilly Media, 2017. 157 p.

3. Lock, A. (2023). ASP.NET Core in Action. 3rd Edition. Manning Publications. 984 p.
4. Esposito, D. (2018). Programming ASP.NET Core (First edition). Microsoft Press. 416 p.
5. Choose an ASP.NET Core web UI. URL: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-7.0>.
6. Model-View-ViewModel (MVVM). URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>.
7. Чому компанії переходять на ASP.NET Core для розробки веб-додатків для вирішення своїх бізнес завдань? [Why companies are switching to ASP.NET Core to develop web applications to solve their business problems?] URL: <https://tqm.com.ua/ua/likbez/ua-articles/chomu-asp-net> [in Ukrainian].
8. Nikonov, O., Skidan, V., Volivach, A., Nadopta, T., Pavlenko, V. (2023). Cloud System of Content Accounting with Access on OC Android and IOS. 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek), Kharkiv, Ukraine, pp. 1002–1005.
9. Skvorchevsky, A. (2022). Increasing the robustness of computer networks by using hybrid centralized-distributed topology. 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, pp. 239–242.
10. Nikonov, O. Ya., Aleksiiev, V. O., Ulko, V. Yu., Seredina, H. I. (2013). Rozroblennia ta vprovadzhennia internet-tehnolohii dlia pidvyshchennia efektyvnosti vykorystannia transportnykh zasobiv [Development and implementation of Internet technologies to increase the efficiency of the use of vehicles]. *Visnyk SevNTU*, № 142, P. 69–72 [in Ukrainian].
11. Nikonov, O. Ya. (2019). Intelektualni kompiuterni tehnolohii rozroblennia transportnykh zasobiv [Intelligent computer technologies for the development of vehicles]. *Visnyk KhNADU*, № 87, P. 49–53 [in Ukrainian].
3. Lock A. ASP.NET Core in Action. 3rd Edition. Manning Publications, 2023. 984 p.
4. Esposito D. Programming ASP.NET Core. First edition. Microsoft Press, 2018. 416 p.
5. Choose an ASP.NET Core web UI. URL: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-7.0>.
6. Model-View-ViewModel (MVVM). URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>.
7. Чому компанії переходять на ASP.NET Core для розробки веб-додатків для вирішення своїх бізнес завдань? URL: <https://tqm.com.ua/ua/likbez/ua-articles/chomu-asp-net>.
8. Nikonov O., Skidan V., Volivach A., Nadopta T., Pavlenko V. Cloud System of Content Accounting with Access on OC Android and IOS. 2023 IEEE 4th KhPI Week on Advanced Technology (KhPIWeek). Kharkiv, Ukraine, 2023. P. 1002–1005.
9. Skvorchevsky A. Increasing the robustness of computer networks by using hybrid centralized-distributed topology. 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT). Lviv, Ukraine, 2022. P. 239–242.
10. Ніконов О. Я., Алексієв В. О., Улько В. Ю., Середіна Г. І. Розроблення та впровадження інтернет-технологій для підвищення ефективності використання транспортних засобів. *Вісник СевНТУ*. 2013. № 142. С. 69–72.
11. Ніконов О. Я. Інтелектуальні комп'ютерні технології розроблення транспортних засобів. *Вісник ХНАДУ*. 2019. № 87. С. 49–53.

SKIDAN VLADYSLAVA

Candidate of Technical Sciences, Associate Professor,
Head of the Department of Information and Computer
Technologies, Kyiv National University of Technologies
and Design, Ukraine

<https://orcid.org/0000-0002-8358-9759>

Scopus Author ID: 57210393405

E-mail: skidan.vv@knuud.edu.ua

NIKONOV OLEH

Doctor of Technical Sciences, Professor,
Professor of the Department of Information and
Computer Technologies, Kyiv National University
of Technologies and Design, Ukraine

<https://orcid.org/0000-0002-8878-4318>

Scopus Author ID: 57210852164

E-mail: nikonov.oy@knuud.edu.ua

VOLIVACH ANTONINA

Candidate of Technical Sciences, Associate Professor
of the Department of Information and Computer
Technologies, Kyiv National University
of Technologies and Design, Ukraine
<https://orcid.org/0000-0002-7119-7774>
Scopus Author ID: 57687762000
E-mail: volivach.ap@knuud.com.ua

PAVLENKO VOLODYMYR

Candidate of Technical Sciences, Associate Professor,
Dean of the Faculty of Mechatronics and Computer
Technologies, Kyiv National University
of Technologies and Design, Ukraine
<https://orcid.org/0000-0003-2163-8508>
Scopus Author ID: 57357228600
E-mail: pavlenko.vm@knuud.edu.ua

SKIDAN V. V., NIKONOV O. J., VOLIVACH A. P., PAVLENKO V. M.

Kyiv National University of Technologies and Design, Ukraine

**RESEARCH OF CLOUD MICROSERVICES BASED
ON ASP.NET CORE TECHNOLOGY**

Goal. Research of cloud microservices based on ASP.NET Core technology and the use of the Model-View-ViewModel MVVM architectural pattern, evaluation of technical advantages.

Methodology. Research of cloud microservices based on ASP.NET Core technology is carried out on the basis of methods and algorithms for the analysis of software systems with the aim of improving their quality, security and productivity.

Results. As a result of the research of cloud microservices based on ASP.NET Core technology, the efficiency of using the MVVM architectural pattern was analyzed. The MVVM architectural pattern allows for the separation of a program's interface, basic presentation and business logic into three distinct classes: the View, which encapsulates the interface and interface logic; the ViewModel, which encapsulates presentation logic and state; the Model, which encapsulates the program's business logic and data. This pattern enables the creation of more scalable and maintainable applications and simplifies the process of testing, support, and application development. Best practices for developing and maintaining microservices in the cloud using ASP.NET Core are explored. On the basis of ASP.NET Core technology and the MVVM architectural pattern, the «Online Gallery» web application for working with photo content has been developed.

Scientific novelty. The use of the MVVM architectural pattern for building cloud microservices and ASP.NET Core technology is proposed. The advantages of using ASP.NET Core in the context of cloud microservices are explored.

Practical value. The research conducted and the results obtained allow us to assess the advantages of implementing ASP.NET Core for cloud microservices, which is important for software architects, developers, and IT companies in general. The obtained results enable informed decisions when designing cloud microservices based on ASP.NET Core technology, which allows you to build more efficient, scalable and secure software systems. The conducted research is the basis for future research and effective implementations in the ever-evolving world of cloud computing and microservices.

Keywords: Web Application; Web Service; Cloud Microservices; ASP.NET Core technology; MVVM architectural pattern.