МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів та архітектурних рішень для підвищення ефективності веб-розробки на платформі .Net
(тема)

Виконав:

Студент 2 курсу групи ІПЗм-22-2

Мітряєв С. С.
(прізвище, ініціали)

Спеціальність 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник доц. Валенда Н. А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ З. В. Дудар
(підпис) (прізвище, ініціали)

2024 р.

Факультет комп’ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:

зав. кафедри_____
(підпис)

«__»_____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Мітряєву Сергію Сергійовичу
(прізвище ім’я по-батькові)

1. Тема роботи «Дослідження методів та архітектурних рішень для підвищення ефективності веб-розробки на платформі .Net»
затверджена наказом університету від «29» березня 2024 р. № 250Ст
2. Термін подання студентом роботи до екзаменаційної комісії «14» червня 2024 р.
3. Вихідні дані до роботи технологія .Net, архітектурні рішення для веб-систем, пояснювальна записка, об’єктно-орієнтоване програмування
4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі і постановка задачі, дослідження видів архітектури проектування веб-систем на мові програмування C#, вивчення можливості їх використання у інформаційній веб-системі, посилання, додатки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	29.03 – 14.04.24	*виконано*
2	Аналіз та вибір методів проектування для дослідження	15.04 – 24.04.24	*виконано*
3	Аналіз та моделювання предметної області	17.04 – 28.04.24	*виконано*
4	Планування дослідження	25.04 – 30.04.24	*виконано*
5	Дослідження обраних методів з використанням фахової літератури	25.04 – 20.05.24	*виконано*
6	Аналіз результатів досліджень та розробка рекомендацій	20.05 – 24.05.24	*виконано*
7	Написання та оформлення статті та тез доповіді	25.04 – 28.04.24	*виконано*
8	Підготовка пояснювальної записки	24.05 – 31.05.24	*виконано*
9	Підготовка презентації та доповіді	01.06 – 02.06.24	*виконано*
10	Нормоконтроль	02.06 – 04.06.24	*виконано*
11	Рецензування	04.06 – 09.06.24	*виконано*
12	Занесення диплома в електронний архів	11.06.2024	*виконано*
13	Попередній захист	11.06.2024	*виконано*
14	Допуск до захисту у зав. кафедри	12.06.2024	*виконано*

Дата видачі « 29 » березня 2024 р.

Студент _____ Мітряєв С. С.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Валенда Н. А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 53 с., 15 рис., 3 табл., 8 джер.

АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ВЕБ-ДОДАТОК, КЛІЄНТ-СЕРВЕР, МЕТОДИ ПРОЕКТУВАННЯ, МІКРО-СЕРВІСИ, МОВА C#, МОНОЛІТНАЯ АРХІТЕКТУРА, МУЛЬТИРІВНЕВІСТЬ, ПЛАТФОРМА .NET, ПОДІЄВО-ОРІЄНТОВАНА АРХІТЕКТУРА, СЕРВІСНО-ОРІЄНТОВАНА АРХІТЕКТУРА.

Об’єктом дослідження є архітектури проектування веб-програмного забезпечення.

Метою роботи є дослідження ефективності різних типів архітектури програмного забезпечення, які використовуються при проектуванні веб-додатків із використанням платформи .Net.

Архітектурою веб-програмного забезпечення є способи структурування програмної системи та абстракція її елементів на певній фазі її роботи. До основних методів розробки відносяться клієнт-серверна, мікро-сервісна, монолітна, мультирівнева та інші види архітектури.

У результаті кваліфікаційної роботи було проаналізовано фахові наукові джерела, що описують такі типи архітектури, як монолітна, мікро-сервісна, мультирівнева, подієво-орієнтована, клієнт-серверна, сервісно-орієнтована. Було проаналізовано їх переваги та недоліки та побудовано математичну модель, яка допомагає у виборі оптимальної архітектури для розробки веб-програмної системи.

CLIENT-SERVER, DESIGN METHODS, EVENT-ORIENTED ARCHITECTURE, LANGUAGE C#, MICROSERVICES, MONOLITHIC ARCHITECTURE, MULTI-TIER, PLATFORM .NET, SOFTWARE ARCHITECTURE, SERVICE-ORIENTED AR ARCHITECTURE, WEB APPLICATION.

The object of research is web software design architectures.

The purpose of the work is to investigate the effectiveness of different types of software architecture used in designing web applications using the .Net platform.

Web software architecture is a way of structuring a software system and abstracting its elements at a certain phase of its operation. The main development methods include client-server, micro-service, monolithic, multi-level, and other types of architecture.

As a result of the qualification work, professional scientific sources describing such types of architecture as monolithic, micro-service, multi-level, event-oriented, client-server, and service-oriented were analyzed. Their advantages and disadvantages were analyzed and a mathematical model was built that helps in choosing the optimal architecture for developing a web software system.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Мітряєв Сергій Сергійович, студент(ка) гр. ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів та архітектурних рішень для підвищення ефективності веб-розробки на платформі .Net», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

COA – Сервісно-орієнтована Архітектура

CGI – Common Gateway Interface

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

MVC – Model–view–controller

MVVM – Model-View-View-Model

REST – Representation State Transfer

ВСТУП

Майже кожна людина у світі у наш час стикається із веб-системами. Вони оточують нас повсюди, коли ми замовляємо продукти, купуємо квитки, перевіряємо банківські акаунти та при інших взаємодіях з інформаційним світом. Це справжнє диво технологій, яке перетворило наше повсякдення, роблячи його більш зручним та доступним. Завдяки веб-системам ми можемо ефективно керувати багатьма аспектами нашого життя безпосередньо через інтернет, забезпечуючи собі швидкість, зручність та ефективність у вирішенні різноманітних завдань.

Але разом з безліччю переваг, які принесли веб-системи, існують і виклики та проблеми, з якими ми стикаємося. Наприклад, зростаюча кількість інформації, яка обробляється веб-системами, підвищує ризики щодо безпеки та конфіденційності даних. Також існує проблема стандартизації та сумісності між різними системами, що може призвести до складнощів у їх використанні та обслуговуванні.

Проте, не дивлячись на ці виклики, веб-системи залишаються ключовим інструментом сучасного світу, який постійно розвивається та вдосконалюється. Із загальним зростанням цифрової грамотності та технологічних можливостей ми можемо очікувати, що веб-системи продовжать забезпечувати нам нові можливості та зручності в майбутньому.

Існує багато різних мов та методів програмування, які дозволяють створювати потужні системи, здатні не лише презентувати інформацію, а й ефективно обробляти та зберігати її. Вибір мови програмування та методів розробки визначається різноманітними факторами, включаючи специфіку проекту, вимоги до продукту та умови роботи [1].

Основними критеріями при розробці програмних систем є легкість користувацької взаємодії, безпека даних, можливість масштабування системи та ціна розробки. Зручність користувацького інтерфейсу є ключовим аспектом, оскільки вона визначає спосіб, яким користувачі будуть взаємодіяти з системою.

Безпека даних є критично важливою для забезпечення конфіденційності та цілісності інформації [1]. Можливість масштабування дозволяє системі розвиватися та працювати ефективно навіть при збільшенні обсягу даних та користувацького трафіку. Ціна розробки визначає економічну доцільність проекту та його відповідність бюджету.

При виборі мов програмування та методів розробки важливо враховувати всі ці фактори та забезпечити баланс між ними для досягнення оптимальних результатів у створенні програмної системи [1].

Також у наш час існує дуже багато різних видів архітектури написання програмних веб-систем, які відображають найновіші технологічні та індустріальні тенденції. Ці архітектури дозволяють оптимізувати процес розробки та забезпечити надійність зберігання даних, водночас забезпечуючи зручність у взаємодії з користувачами. Вони відображають сучасні підходи до розробки програмного забезпечення та використовують передові методи та технології для досягнення оптимальних результатів [1].

Незважаючи на те, що кожна з цих архітектур має свої переваги, важливо розуміти, що вони також мають свої недоліки [1].

Для обрання видів архітектури пропонується використати математичні моделі для прийняття рішень такі, як метод лінійної згортки з коефіцієнтами, оптимальні множини Парето та інші [2]. У даній роботі також наведено опис використаних методів з відображенням їх переваг та недоліків.

Отже, метою даної роботи є вирішення проблеми обрання необхідної методики та архітектури розробки веб-систем, яка зможе надати максимальну кількість переваг та мінімальну кількість недоліків. Для цього будуть розглянуті основні види архітектури проектування програмного забезпечення та за допомогою математичних методів буде обрано найбільш оптимальні для написання інформаційної веб-системи для взаємодії з користувачами та обробки, зберігання та відображення інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

1.1.1 Розвиток предметної галузі

Звичайно, історія розвитку програмних веб-систем справді вражає своєю стрімкістю, особливо з початку 1990-х років. У ті дні веб-сервери використовували Common Gateway Interface для виклику зовнішніх програм та генерації веб-сторінок у відповідь на запити користувачів [3].

Однак процес CGI мав свої недоліки, зокрема, кожен запит на сервер викликав новий процес CGI. Це призводило до великого навантаження на сервер, що здійснювалося через велику кількість створюваних процесів, що в свою чергу зменшувало продуктивність системи [3].

Ці обмеження і проблеми спонукали розробників до пошуку нових методів та технологій для покращення продуктивності та ефективності веб-систем. Такі пошуки привели до з'яви нових підходів, які були спрямовані на оптимізацію роботи веб-серверів та зменшення навантаження на них [3].

Насправді, ці виклики стимулювали розвиток технологій, таких як модель подій, асинхронні запити та інші інноваційні підходи, які дозволили збільшити продуктивність та масштабованість веб-систем. Таким чином, згадувана еволюція від CGI до сучасних методів розробки веб-систем є яскравим прикладом того, як технології веб-розробки постійно вдосконалюються та адаптуються до змін у вимогах та технологічних можливостях. [3].

З'явилися серверні скриптові мови, які дозволяли розробникам вбудовувати динамічний контент без необхідності викликати новий процес для кожного запиту. Зменшення завдань, які виконувались на сервері, завдяки поширенню використання JavaScript для обробки подій на боці клієнта [3].

Впровадження технології AJAX дозволило веб-додаткам асинхронно взаємодіяти з сервером без перезавантаження сторінки. З'явилися односторінкові додатки, які використовують JavaScript для динамічного оновлення контенту на одній сторінці [1].

Збільшення складності веб-додатків призвело до використання мікро-сервісної архітектури, де функціональність розбивається на невеликі та незалежні мікро-сервіси. Розвиток контейнеризації, такі як Docker та розгорткових систем, наприклад, Kubernetes, полегшив управління мікро-сервісами [1].

Постійний розвиток SPA і Progressive Web Applications дозволяє створювати більш інтерактивні та ефективні веб-додатки. Архітектура JAMstack дозволяє створювати веб-сайти та додатки, які є більш масштабованими та швидшими завдяки використанню попередньої генерації контенту. Збільшилося використання серверних функцій для автоматизації і управління ресурсами серверів. Розвиток інструментів для автоматизації та контролю серверного менеджменту для полегшення конфігурації та розгортання інфраструктури [5].

Розвиток архітектури веб-систем відображає стійке прагнення до ефективності, швидкості та масштабованості веб-додатків. Технологічні зміни та новаторські підходи сприяли створенню більш сучасних, гнучких та потужних систем.

1.1.2 Аналіз існуючих архітектур проектування

В екосистемі .Net існує розмаїття архітектурних підходів для проектування веб-застосунків. Серед найпоширеніших варіантів варто відзначити Model-View-Controller, де додаток розбивається на три основні складові: модель, представлення та контролер. Цей підхід підтримується такими відомими фреймворками, як ASP.NET MVC і ASP.NET Core MVC, що дозволяють ефективно організувати роботу з веб-додатками [6].

Також важливим є підхід Model-View-View-Model (MVVM), особливо у контексті розробки односторінкових додатків. MVVM розділяє додаток на три компоненти: модель, відображення та модель представлення. Цей підхід добре поєднується з такими фреймворками, як Angular, Knockout.js та Vue.js, що надають інструменти для зручної реалізації інтерактивного користувацького інтерфейсу та управління даними на клієнтській стороні.

Кожен з цих підходів має свої переваги та відповідає різним потребам та сценаріям розробки веб-додатків у світі .Net. [4].

Blazor представляє інноваційний підхід до розробки веб-застосунків, дозволяючи використовувати мову програмування C# та платформу .Net як на стороні клієнта, так і на стороні сервера. Цей підхід відкриває нові можливості для розробників, що працюють у екосистемі .Net, спрощуючи розробку та підтримку веб-додатків [6].

RESTful API є важливим компонентом в архітектурі Blazor, надаючи засіб для взаємодії між клієнтом та сервером за допомогою стандартних HTTP-запитів. Це дозволяє реалізувати обмін даними та функціональні можливості між клієнтською та серверною частинами додатка. Для реалізації RESTful API в Blazor можна використовувати такі інструменти, як ASP.NET Web API або ASP.NET Core Web API, які надають зручний спосіб створення та підтримки API в екосистемі .Net.

Blazor разом з RESTful API відкриває широкі можливості для розробки високопродуктивних та масштабованих веб-додатків з використанням мови програмування C# та інструментів .Net. Цей підхід дозволяє розробникам ефективно використовувати свої знання та досвід у сфері .Net для створення сучасних веб-застосунків. [6].

Мікро-сервісна архітектура стала ключовим інструментом у сучасній розробці програмного забезпечення, завдяки своїй здатності забезпечувати гнучкість, масштабованість і розширюваність систем. Цей підхід перетворює великі та складні додатки на набір менших, незалежних компонентів, що спрощує їх розробку, тестування та підтримку.

Однією з ключових переваг мікро-сервісної архітектури є можливість незалежного розгортання та масштабування кожного мікро-сервісу. Це означає, що різні частини системи можуть бути розгорнуті на різних серверах або навіть у різних хмарних сервісах, що дозволяє ефективно використовувати ресурси та забезпечувати високу доступність.

Крім того, мікро-сервіси можуть бути розроблені, випробувані і впроваджені окремо один від одного, що полегшує розробку та підтримку. Кожен мікро-сервіс

може виконувати конкретні функції або обслуговувати окремі частини бізнес-логіки, що дозволяє командам розробників працювати над окремими компонентами системи незалежно один від одного.

У цілому, мікро-сервісна архітектура сприяє швидкій реакції на зміни в бізнес-вимогах, забезпечує високу гнучкість та прискорює процес впровадження нового функціоналу. Це робить її незамінною для розробки та підтримки сучасних програмних продуктів. [5].

У сучасних системах, окрім мікро-сервісної архітектури, все більшою популярністю користується підхід до серверного програмування, який використовує функції для автономного виконання завдань. Це дозволяє окремим функціям або блокам коду працювати незалежно від централізованого сервера, що призводить до збільшення автономності та стійкості системи [5].

Завдяки цьому підходу, система може ефективно використовувати обмежені ресурси, оскільки окремі функції можуть бути активовані лише тоді, коли потрібно, та виконувати свою роботу незалежно одна від одної. Це дозволяє оптимізувати використання обчислювальних потужностей та мінімізувати витрати [5].

Крім того, використання функцій для автономного виконання завдань забезпечує високий рівень стійкості системи до відмов. Навіть у випадку, коли одна з функцій виявляє несправність або відмову, інші функції можуть продовжувати працювати без перебоїв, що дозволяє системі продовжувати нормальну роботу.

У цілому, використання функцій для автономного виконання завдань сприяє створенню більш стійких, ефективних та ресурсозберігаючих систем, що відповідають сучасним вимогам до надійності та продуктивності. [5].

Разом ці архітектурні підходи дозволяють створювати системи, які легко масштабувати, гнучкі у реагуванні на зміни та стійкі до відмов, що є ключовими вимогами для сучасних веб-додатків та сервісів. [5].

У світі веб-розробки архітектурний рух JAMstack, що базується на принципах JavaScript, API та маркапу, набуває все більшої популярності. Один з ключових компонентів JAMstack - це використання різноманітних фреймворків, серед яких особливе місце посідає Next.js [6].

Next.js відзначається не лише своєю ефективністю та швидкодією, але й широким спектром можливостей для реалізації різноманітних веб-додатків. Він дозволяє розробникам створювати як прості статичні сайти, так і складні динамічні веб-застосунки, забезпечуючи високий рівень продуктивності та масштабованості [6].

У контексті JAMstack, фреймворки, такі як Next.js, працюють у єдиному екосистемі з іншими інструментами, такими як статичні генератори сайтів та сервіси для роботи з API. Це дозволяє розробникам швидко та ефективно реалізовувати веб-додатки, що відповідають вимогам сучасного веб-середовища.

З використанням фреймворків, таких як Next.js, в рамках JAMstack розробники отримують потужний інструментарій для створення продуктивних та масштабованих веб-додатків, що задовольняють найвищі стандарти якості та відповідають потребам сучасного веб-світу. [6].

1.2 Формування мети

Метою даної роботи є дослідження вибору оптимального варіанту архітектури розробки програмної інформаційної веб-системи на платформі .Net. Увагу у даній роботі планується зосередити на таких варіантах архітектури, як: монолітна, клієнт-серверна, сервісно-орієнтована, мікро-сервісна, мультирівнева та подієво-орієнтована архітектури.

Конкретні питання, що досліджуються включають у себе:

- наскільки безпечно зберігати дані у додатках з даними видами архітектури;
- наскільки гнучкою є архітектура;
- чи є можливість масштабування системи;
- продуктивність програмної системи;
- наскільки легко обслуговувати дане архітектурне рішення.

Мета дослідницької роботи полягає у розгляді та аналізі різних архітектурних підходів до проектування програмного забезпечення з метою знаходження оптимальних рішень для розробки програмної інформаційної веб-системи.

Основна увага при цьому зосереджується на забезпеченні безперервної та надійної роботи системи, враховуючи усі необхідні критерії.

Дослідження включає аналіз різноманітних архітектурних підходів, таких як мікро-сервісна архітектура, монолітна архітектура, серверно-клієнтська архітектура та інші. Кожен з цих підходів має свої переваги та недоліки, які варто розглянути у контексті розробки веб-систем на платформі .Net.

Мікро-сервісна архітектура може забезпечити більшу гнучкість та масштабованість системи за рахунок розбиття функціональності на незалежні сервіси. Монолітна архітектура, з іншого боку, може бути простішою у впровадженні та управлінні, але менш гнучкою у зміні та масштабуванні. Серверно-клієнтська архітектура може забезпечити більшу ефективність у використанні ресурсів та кращу реактивність, але може вимагати більшої кількості обміну даними між клієнтом та сервером.

Важливо провести глибокий аналіз кожного з цих підходів і знайти оптимальне рішення, яке відповідатиме вимогам та потребам конкретного проекту з розробки веб-системи на платформі .Net. Такий підхід дозволить забезпечити успішну реалізацію проекту та досягнення поставлених цілей. [4].

У результаті дослідження ми отримаємо детальний огляд та порівняння різних видів архітектури розробки та проектування веб інформаційної програмної системи із використанням платформи для розробки та керування програмного забезпечення .Net. Це дослідження може надати цінну інформацію про переваги та недоліки кожної архітектурної альтернативи, враховуючи такі критерії, як продуктивність, масштабованість, надійність, витрати на реалізацію, та інші важливі фактори.

Також у процесі дослідження буде виділено можливості покращення процесів розробки та проектування програмних систем для різних класів інформаційних систем, які використовують платформу .Net. Це дозволить виявити оптимальні підходи та найкращі практики для підвищення таких характеристик, як безпека, швидкодія, легкість підтримки та розвитку програмного забезпечення. У сучасних умовах ці аспекти є критичними для забезпечення високої якості та

конкурентоспроможності інформаційних систем. Результати дослідження сприятимуть:

- підвищенню безпеки: Розробка рекомендацій та стандартів для покращення захисту даних та забезпечення безпеки веб-систем, розроблених на платформі .Net.
- покращенню швидкодії: Виявлення ефективних методів оптимізації продуктивності, що дозволить створювати швидкі та ефективні веб-додатки.
- забезпеченню легкості підтримки: Розробка підходів, які спрощують процес підтримки та оновлення систем, знижуючи витрати на їх обслуговування.
- спрощенню розробки: Впровадження кращих практик, що спрощують процес розробки, знижують складність та підвищують ефективність роботи розробників.

Ці досягнення є необхідними вимогами при розробці інформаційного програмного забезпечення в наш час, оскільки вони сприяють підвищенню якості продукту та його конкурентоспроможності на ринку. Забезпечення надійності, ефективності та зручності взаємодії з користувачами стає визначальними чинниками успіху програмного забезпечення в сучасному інформаційному середовищі.

Таким чином, наше дослідження стане вагомим внеском у розвиток ефективних методів і практик розробки програмного забезпечення на платформі .Net. Розробка нових технік, оптимізація існуючих підходів та виявлення кращих практик дозволять підвищити якість та продуктивність програмних продуктів, створених на базі .Net. Це буде корисно як для розробників, які зможуть використовувати отримані знання для покращення своїх проектів, так і для кінцевих користувачів, які отримають доступ до якісного, надійного та зручного у використанні програмного забезпечення. Такий підхід сприятиме збільшенню конкурентоспроможності компаній, які використовують розроблені нашим дослідженням технології, на ринку програмного забезпечення.

2 МАТЕМАТИЧНЕ ПРЕДСТАВЛЕННЯ

2.1 Багатокритеріальна задача вибору варіанту архітектури

Першим кроком у описі задачі багатокритеріального вибору варіанту архітектури веб-системи є визначення критеріїв, які відображають вимоги та обмеження проекту. Основні критерії, які можуть мати вплив на вибір архітектури, можуть бути наступними:

- вартість: загальна вартість розробки та підтримки проекту за обраною архітектурою;
- масштабованість: можливість легкого внесення змін у кількість користувачів, що одночасно можуть користуватися системою;
- місце зберігання даних: можливість зберігати дані віддалено або локально та створювати резервні копії;
- надійність: можливість запобігати критичним помилкам та відмовам системи та швидко відновлювати її роботу;
- гнучкість системи: можливість легко вносити зміни у систему без необхідності у значних змінах системи;
- безпека: підтримка системою захисту персональних даних від витоків.

Окрім даних критерії, ще можна розглядати інші, менш суттєві критерії, такі, як легкість користуванням, продуктивність та інші.

Отже, задача вибору архітектури для розробки інформаційної програмної веб-системи відноситься до класу багатокритеріальної задачі вибору, де кожен критерій має певну вагу, яка відображає його важливість для проекту. Завдання даного дослідження полягає у тому, щоб обрати архітектуру розробки програмного забезпечення, яка забезпечує найкраще поєднання даних критеріїв, що будуть задовольняти вимогам та обмеженням даного проекту.

Існують різні способи розв’язання задач багатокритеріального вибору. В рамках даного дослідження, ми розглянемо метод, який дозволяє вирішити дану задачу, враховуючи найважливіші критерії розробки програмної веб-системи та

враховуючи їх вагу [2]. Для цього завдання ми будемо використовувати метод адитивної лінійної згортки із ваговими коефіцієнтами .

Ідея даного метода адитивної лінійної згортки із ваговими коефіцієнтами полягає у тому, щоб ефективно оцінити та вибрати оптимальний варіант з набору альтернатив, приділяючи вагомий внесок кожному критерію відповідно до його важливості для прийняття рішення.

Кожен критерій має свій ваговий коефіцієнт, який відображає його важливість у прийнятті рішення. Ці вагові коефіцієнти можуть визначатися експертами або іншими методами, які враховують пріоритети та представлення індивідуальних критеріїв [2].

Кожен варіант розглядається в рамках кожного критерію, і його результат згортається з ваговим коефіцієнтом. Сума цих згорток створює загальний скорочений показник ефективності для кожного варіанту. На основі отриманих скорочених показників ефективності можна здійснити порівняння між альтернативами. Варіант з найвищим сумарним значенням є оптимальним з точки зору заданих вагових коефіцієнтів [2].

Метод адитивної лінійної згортки дозволяє ефективно працювати з великою кількістю даних та критеріїв, спрощуючи процес прийняття рішення шляхом врахування їхнього взаємного впливу.

Математично даний метод може бути виражений наступним чином за формулою 2.1 для обчислення загального показника ефективності S_i для кожної альтернативи i:

$$S_i = \sum_{j=1}^{n} \omega_j \cdot x_{ij}, \tag{2.1}$$

де n – кількість критеріїв;

x_{ij} – значення i-го критерію для j-тої альтернативи;

ω_j – ваговий коефіцієнт для j-тої альтернативи.

Наведена формула 2.1 виражає суму добутків значень кожного критерію для

конкретної альтернативи на його відповідний ваговий коефіцієнт. Отримані суми можуть бути порівняні між альтернативами, і альтернатива з найвищим значенням загального показника ефективності вважатиметься оптимальною з урахуванням вагових коефіцієнтів [2].

Хоча метод адитивної лінійної згортки із ваговими коефіцієнтами є ефективним і часто використовується для багатокритеріального вибору, він також має свої основні недоліки:

– лінійність: метод передбачає лінійну залежність між критеріями та їхніми ваговими коефіцієнтами. Однак у деяких випадках взаємозв'язок між критеріями може бути складнішим, і лінійна модель не здатна відобразити всі аспекти взаємодії;
– суб’єктивність вагових коефіцієнтів: присвоєння вагових коефіцієнтів є суб'єктивним процесом і може бути важким завданням. Вибір неправильних вагових коефіцієнтів може спричинити виникнення відхилень у результатах;
– відсутність урахування невизначеності: метод не враховує невизначеність або непевність у взаємозв'язках між критеріями та їхніми значеннями;
– важкість роботи з більшою кількістю критеріїв: при великій кількості критеріїв може виникнути складність у визначенні адекватних вагових коефіцієнтів та аналізу взаємодії між критеріями.

У нашому випадку, зважаючи на вибір предметної галузі та особливості реалізації програмного забезпечення, ми можемо знехтувати цими недоліками та обрати метод адитивної лінійної згортки з ваговими коефіцієнтами для вирішення проблем даного дослідження.

Наступним кроком необхідно обрати критерії, які будуть описувати наші альтернативи вибору архітектури розробки веб програмного забезпечення на платформі .Net, враховуючи поставлені обмеження. Після визначення критеріїв, ми зможемо провести об'єктивну оцінку кожної альтернативи та визначити, яка архітектура найбільш відповідає нашим потребам та обмеженням проекту.

2.2 Архітектури розробки веб програмного забезпечення

Існують багато різних видів архітектури програмно забезпечення, які можна використовувати для розробки веб-застосунків на платформі .Net. Для даного дослідження будуть обрані найбільш популярні [1]. Але по-перше потрібно виділити характеристики видів архітектури, які будемо використовувати при багатокритеріальному аналізі:

- можливість масштабування: один із основних критеріїв для розробки веб-системи. В наш час кількість користувачів систем постійно збільшується і потрібно мати можливість збільшувати пропускну здатність веб-системи;
- гнучкість системи: кожна система повинна мати можливість легкої зміни її функцій та вигляду для того, щоб задовольняти постійні зміни у вимогах користувачів без необхідності повного переписування коду програми;
- легкість в обслуговуванні: система повинна мати можливість легкого обслуговування без необхідності довгого навчання нових розробників;
- продуктивність: система повинна легко справлятися з великим навантаженням від користувачів;
- безпека: дуже важливий критерій обрання архітектури, особливо у наш час, коли персональні дані користувачів повинні бути захищеними.

Дані характеристики можуть відрізнятися у кожному з типів архітектури, та саме завдяки їм ми можемо обрати ту, що буде найбільш оптимальною для розробки веб програмного забезпечення, що буде задовольняти вимогам та обмеженням проекту.

2.2.1 Монолітна архітектура

Монолітна архітектура є традиційним підходом до розробки веб-систем і полягає в тому, що всі компоненти додатку розташовані в одному монолітному кодовому базисі та виконуються як один великий модуль [1]. Даний модуль не розділяється на підрівні та має як переваги, так і значні недоліки, які будуть розглянуті далі. Цей підхід протиставляється мікро-сервісній архітектурі.

Основні характеристики даної моделі наведено нижче:

– єдиний кодовий базис: усі функціональні елементи додатку розташовані в одному кодовому базисі. Це спрощує розробку та обслуговування;
– єдиний процес: всі компоненти додатку виконуються в одному процесі на сервері. Інколи це означає, що додаток працює в одному серверному контейнері або процесі;
– просте масштабування: масштабування в основному виконується вертикально (додавання ресурсів до одного сервера) або горизонтально (запуск кількох копій всього додатку);
– прості розробка та розгортання: розробка, тестування і розгортання можуть бути простішими порівняно з іншими архітектурними підходами;
– спільне зберігання даних: всі дані зберігаються у спільній базі даних. Зазвичай це одна база даних, яка обслуговує всі компоненти додатку;
– спільна безпека: часто використовується спільна система безпеки та ідентифікації для всіх компонентів.

Однако, монолітна архітектура має свої недоліки, особливо в контексті масштабованості та гнучкості. З ростом об'єму коду і складності системи, розробка та обслуговування може стати важче, а масштабованість може бути обмеженою. Тому в останні роки деякі організації переходять до більш гнучких архітектурних підходів, таких як мікро-сервісна, клієнт-серверна або подієво-орієнтована архітектури [1].

2.2.2 Клієнт-серверна архітектура

Клієнт-серверна архітектура є однією з основних типів архітектури для розробки веб-систем і забезпечує розділення функцій між клієнтською і серверною частинами додатку [1,4]. Основні характеристики даної моделі наведено нижче:

– асинхронна взаємодія: в деяких випадках може бути використана асинхронна взаємодія, наприклад, за допомогою технологій WebSockets, щоб забезпечити більш ефективну взаємодію в реальному часі;

- протокол взаємодії: клієнт та сервер взаємодіють за допомогою одного з протоколів передачі даних, як наприклад HTTP;
- клієнтська частина: клієнтська сторона включає в себе інтерфейс користувача та логіку, яка виконується на стороні користувача;
- серверна частина: серверна сторона обробляє бізнес-логіку, взаємодіє з базою даних, і відправляє необхідні дані на клієнтську сторону;
- розподілення обов'язків: розділення функціональності між клієнтською та серверною стороною дозволяє добре організовані та підтримувані додатки;
- модель запит-відповідь: клієнт відправляє запити до сервера, а сервер обробляє ці запити і надсилає відповіді.

Однією з проблем є залежність від мережі для постійного підключення до сервера, що може викликати обмеження у доступі до даних при відсутності мережі або неналежній якості підключення. Збільшення кількості клієнтів може також призвести до обмежень у масштабованості серверної сторони, і потрібність у більш потужних серверах для обробки більшої кількості запитів. Незважаючи на ці недоліки, клієнт-серверна архітектура залишається ефективною та широко використовуваною [4].

2.2.3 Мультирівнева архітектура

Мультирівнева архітектура розробки веб-систем - це підхід, при якому функціональність додатку розподілена між різними рівнями або шарами. Кожен рівень має визначену функціональність і взаємодіє тільки з певними іншими рівнями, що забезпечує модульність, гнучкість та підтримку розширюваності [1, 4]. Основні рівні даної моделі наведено нижче:

- рівень представлення: цей рівень відповідає за представлення даних користувачам та обробку їхніх запитів. Включає інтерфейс користувача та логіку відображення;
- рівень логіки: цей рівень відповідає за бізнес-логіку додатку. Включає опрацювання даних, виконання бізнес-правил та обчислення;

- рівень даних: цей рівень відповідає за взаємодію з базою даних та забезпечення доступу до даних. Включає запити до бази даних, збереження та отримання даних;
- рівень інфраструктури: цей рівень забезпечує інфраструктуру та базові сервіси для додатку. Може включати сервіси авторизації, кешування, роботу з файлами та інші базові функціональності;
- рівень безпеки: цей рівень відповідає за забезпечення безпеки додатку, включаючи автентифікацію, авторизацію та захист від атак;
- сервісний рівень: може включати різні служби, такі як веб-сервіси або мікро-сервіси, які надають конкретні функції для внутрішнього чи зовнішнього використання;
- рівень комунікацій: може включати різні засоби комунікації між рівнями, такі як месенджери, шини подій чи протоколи взаємодії.

Мультирівнева архітектура дозволяє досягти високої модульності, гнучкості та легкості розширення веб-систем. Розділення функціональності на рівні сприяє легшому тестуванню, підтримці та розвитку проекту [4].

Незважаючи на переваги, мультирівнева архітектура також супроводжується певними недоліками. Збільшення кількості рівнів може призводити до збільшення складності системи та затримок відповіді. Передача даних між рівнями може викликати додаткове перевантаження, а спільні компоненти можуть стати джерелом складнощів у підтримці та модифікації системи [1, 4].

2.2.4 Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура дозволяє досягти високої модульності, гнучкості та легкості розширення веб-систем. Розділення функціональності на рівні сприяє легшому тестуванню, підтримці та розвитку проекту [1]. Основні характеристики даної моделі наведено нижче:

- сервіси: система розбивається на окремі сервіси, які представляють конкретні функціональні частини додатку. Кожен сервіс виконує конкретну задачу та може взаємодіяти з іншими сервісами;

- взаємодія через протоколи: для взаємодії між сервісами використовуються стандартні протоколи, такі як HTTP/REST або Message Queues, що спрощує комунікацію та інтеграцію;
- незалежність: сервіси можуть бути розгорнуті та функціонувати незалежно. Це дозволяє їм бути локалізованими та масштабованими окремо від інших частин системи;
- безпека: забезпечення безпеки в сервісно-орієнтованій архітектурі вимагає дотримання стандартів автентифікації, авторизації та шифрування при обміні даними;
- можливість керування станами: кожен сервіс зберігає власний стан, включаючи дані та логіку. Це підтримує принцип ізоляції та незалежності.

Зі збільшенням кількості сервісів у системі ускладняються управління, моніторинг та координація цих сервісів, що може призвести до збільшення витрат на ресурси. Забезпечення синхронізації та однорідності даних між різними сервісами може стати складною задачею, особливо при великій кількості взаємодій.

З мережевого погляду велика кількість взаємодій також може призводити до збільшення обсягу мережевого трафіку та витрат на мережеві ресурси. Забезпечення безпеки у великих та розподілених системах СОА може вимагати складних заходів для управління доступом та забезпечення конфіденційності даних.

Розробка та підтримка інфраструктури для управління багатьма сервісами може вимагати додаткових зусиль та ресурсів. Вибір різних технологій для різних сервісів може викликати проблеми сумісності та інтеграції. Налагодження та тестування системи на основі СОА можуть бути складнішими порівняно з монолітними системами через розподіленість та різноманітність компонентів [1].

2.2.5 Подієво-орієнтована архітектура

Подієво-орієнтована архітектура є підходом до розробки веб-систем, в основі якого лежить обробка та обмін подіями між компонентами системи [4]. У цій архітектурі основні дії та комунікації відбуваються через обробку подій, що робить

систему гнучкою та здатною до асинхронної обробки даних. Основні характеристики даної моделі наведено нижче:

- асинхронність: система працює асинхронно, реагуючи на події під час їхнього виникнення. Це дозволяє ефективно управляти ресурсами та реагувати на події в реальному часі;
- події: всі події, які відбуваються в системі, розглядаються як основні елементи. Це може включати дії користувачів, зміни стану системи, або будь-які інші події, які можуть викликати реакцію системи;
- локальна обробка подій: кожен компонент може обробляти події локально та генерувати нові події для сприяння взаємодії та реакції системи;
- масштабованість: подієво-орієнтована архітектура може бути масштабованою, оскільки компоненти можуть функціонувати незалежно один від одного, обмінюючи події за потреби;
- стійкість до збоїв: система може бути стійкою до збоїв, оскільки вона може продовжувати роботу після виникнення помилок у частинах системи, завдяки асинхронній обробці подій.

Серед основних викликів ПОА можна виділити складність розробки та розуміння системи для тих, хто не знайомий з цією архітектурою. Зі збільшенням кількості подій у системі може зростати складність управління станом системи та виявлення причин помилок [1].

Також, велика кількість подій та взаємодій може впливати на продуктивність системи та призводити до складнощів у виявленні та усуненні проблем. Використання асинхронності може викликати труднощі у налагодженні та тестуванні, особливо для розробників, не звиклих до асинхронного програмування [4].

Потенційні проблеми також можуть виникнути у сфері забезпечення безпеки, так як зі збільшенням числа подій та взаємодій може зростати ризик недостатньої аутентифікації та авторизації в асинхронному середовищі. Крім того, системна асинхронність може викликати проблеми у виявленні та виправленні помилок, що може ускладнювати роботу розробників [4].

2.2.5 Мікро-сервісна архітектура

Мікро-сервісна архітектура є підходом до розробки веб-систем, де програмний продукт розбивається на невеликі, автономні та незалежні компоненти, відомі як мікро-сервіси [1, 4]. Основні характеристики даної моделі наведено нижче:

- автономність мікро-сервісів: кожен мікро-сервіс є автономним, що дозволяє йому функціонувати незалежно від інших компонентів системи. Це сприяє гнучкості та розширюваності [5];
- масштабованість: мікро-сервіси можуть розгортатися та функціонувати незалежно один від одного, дозволяючи розподілену розробку та масштабованість [1,5];
- крос-функціональність: кожен мікро-сервіс може бути розроблений та управлятися окремою крос-функціональною командою, що сприяє швидкому розвитку та реагуванню на зміни [5];
- API-взаємодія: мікро-сервіси взаємодіють між собою за допомогою API, що дозволяє їм взаємодіяти та обмінюватися даними [1,5];
- ізоляція: мікро-сервіси можуть бути відтворюваними та ізольованими, забезпечуючи стійкість до збоїв та підтримуючи високий рівень доступності [1];
- авторизація: кожен мікро-сервіс може бути моніторингом та авторизацією окремо, що полегшує виявлення та виправлення помилок [5];
- безпека: забезпечення безпеки в розподіленому середовищі може вимагати додаткових заходів, особливо при обміні чутливою інформацією між сервісами [5].

Разом із численними перевагами, мікро-сервісна архітектура має свої власні недоліки. Управління великою кількістю мікро-сервісів може бути важким завданням, вимагаючи ефективної системи моніторингу. Збільшення обсягу мережевого трафіку та складність тестування і налагодження також можуть бути викликами [1, 6].

Додатково, підтримка мікро-сервісів може вимагати додаткової інфраструктури для моніторингу, балансування навантаження та керування конфігурацією. Забезпечення безпеки в розподіленому середовищі і управління транзакціями також може виявитися складним завданням [1].

2.3 Шкали для оцінювання альтернатив

Для кожного критерію потрібно обрати шкалу для подальшого оцінювання методом адитивної лінійної згортки із коефіцієнтами. Пропонується використовувати якісні шкали з переводом результатів у бали від 1 до 5.

2.3.1 Вимірювання масштабованості

Потрібно розробити 5 рівнів для оцінювання масштабованості, які наведено нижче:

- дуже низька (1 бал): архітектура не здатна ефективно масштабуватись. Ймовірно, це монолітна програма або інша архітектура, яка має обмежені можливості горизонтального або вертикального масштабування;
- низька (2 бали): є деякі можливості масштабування, але вони обмежені або вимагають значних зусиль. Це може бути архітектура з обмеженою горизонтальною масштабованістю;
- середня (3 бали): архітектура забезпечує задовільні можливості масштабування. Її можна ефективно масштабувати залежно від потреб проекту;
- висока (4 бали): архітектура має гарні можливості масштабування. Горизонтальне та вертикальне масштабування виконуються з невеликими зусиллями;
- дуже висока (5 балів): це архітектура, яка надає відмінні та легко масштабовані рішення. Горизонтальне та вертикальне масштабування виконуються легко та ефективно.

2.3.2 Вимірювання гнучкості

Потрібно розробити 5 рівнів для оцінювання гнучкості, які наведено нижче:

- дуже низька (1 бал): архітектура має обмежені можливості для внесення змін. Зміни потребують значних зусиль і можуть бути складними;
- низька (2 бали): є деяка гнучкість, але вона обмежена. Внесення змін може бути скрутним;
- середня (3 бали): архітектура надає задовільні можливості для внесення змін. Можна вносити зміни з деякими труднощами;
- висока (4 бали): архітектура має гарну гнучкість. Внесення змін зазвичай легко і вимагає значних змін у структурі системи;
- дуже висока (5 балів): це архітектура, яка надає чудову гнучкість. Система легко адаптується до змін без істотного на її структуру.

2.3.2 Вимірювання легкості обслуговування

Потрібно розробити 5 рівнів для оцінювання легкості обслуговування, які наведено нижче:

- дуже низька (1 бал): обслуговування архітектури потребує значних зусиль. Можливо, система схильна до частих збоїв або складності у виявленні та виправленні проблем;
- низька (2 бали): обслуговування архітектури важко. Регулярні збої та проблеми в управлінні;
- середня (3 бали): архітектура має задовільну легкість в обслуговуванні. Обслуговування вимагає певних зусиль, але не проблематичне;
- висока (4 бали): обслуговування архітектури легко. Можливості моніторингу та управління забезпечуються на високому рівні;
- дуже висока (5 балів): це архітектура, яка забезпечує відмінну легкість обслуговування. Система стабільна, і проблеми швидко виявляються та усуваються.

2.3.4 Вимірювання продуктивності

Потрібно розробити 5 рівнів для оцінювання продуктивності, які наведено нижче:

- дуже низька (1 бал): низька продуктивність. Тривалі часи відповіді та часті збої;
- низька (2 бали): продуктивність нижче за середню. Істотні проблеми із продуктивністю;
- середня (3 бали): усереднена продуктивність. Система працює задовільно, але може бути оптимізована;
- висока (4 бали): гарна продуктивність. Низькі часи відповіді та стабільна робота системи;
- дуже висока (5 балів): відмінна продуктивність. Система ефективна.

2.3.5 Вимірювання безпеки

Потрібно розробити 3 рівні для оцінювання безпеки, які наведено нижче:

- дуже низька (1 бал): рівень безпеки нижче середнього. Суттєві вразливості потребують уваги;
- низька (2 бали): задовільний рівень безпеки. Деякі вразливості можуть бути виявлені та усунені;
- середня (3 бали): високий рівень безпеки. Система захищена від багатьох загроз;

2.4 Оцінювання альтернатив за критеріями

Після створення шкал вимірювання критеріїв, зважаючи на доступну інформацію про альтернативи, ми отримуємо можливість оцінити їх за даними шкалами. Для цього кожну із альтернатив необхідно оцінити за кожною із шкал за заданою кількістю балів, які було присвоєно базуючись на аналізі даних із професійних наукових джерел (див. табл. 2.1).

Таблиця 2.1 – Оцінювання альтернатив за шкалами (таблиця виконана самостійно)

	Масштабованість	Гнучкість	Легкість в обслуговуванні	Продуктивність	Безпека
Монолітна	2	2	3	3	1
Клієнт-сервер	3	3	4	5	2
Мультирівнева	3	4	4	2	3
Сервісно-орієнтована	4	5	4	3	2
Мікро-сервісна	5	3	4	4	2
Подієво-орієнтована	4	4	5	2	2

Використовуючи дані із вищенаведеної таблиці, ми можемо провести розрахунок корисності для кожної з альтернатив. Корисність альтернативи визначається як числовий показник, що відображає ступінь відповідності альтернативи вимогам чи критеріям, встановленим при прийнятті рішення.

Для цього можна використовувати різні методи оцінки, такі як вагова сума балів або метод аналізу ієрархій. Враховуючи важливість кожного критерію, можна призначити вагу кожному з них і визначити, наскільки кожна альтернатива задовольняє ці критерії.

Розрахунок корисності дозволяє об'єктивно оцінити та порівняти різні альтернативи, виходячи з конкретних вимог та обмежень проекту. Цей підхід забезпечує систематизований і прозорий процес прийняття рішень, що мінімізує суб'єктивність та сприяє вибору найкращої архітектури для розробки веб-системи. Такий підхід є важливим етапом у процесі розробки програмного забезпечення, оскільки дозволяє раціонально використовувати ресурси та досягати максимальних результатів у реалізації проекту.

3 ПРОВЕДЕННЯ РОЗРАХУНКІВ КОРИСНОСТІ АЛЬТЕРНАТИВ

3.1 Приведення шкал до принципу оптимальності “За максимумом”

Дані шкали немає необхідності приводити до одного принципу оптимальності, так як вони вже приведені за принципом “за максимумом.” Усі шкали мають оцінки відповідні до внесків до методу вибору оптимальності, тобто менший бал — менший внесок до вибору альтернативи.

3.2 Обрання підмножини Парето

Важливим етапом пошуку оптимального варіанту є обрання підмножини альтернатив за принципом Парето. Обрання підмножини Парето – важливий етап у багатокритеріальному виборі, спрямований на визначення оптимальних рішень, які не мають собі рівних за всіма критеріями. Процес починається з оцінки рішень за різними критеріями та побудови матриці порівнянь [7,8]. Визначення відносної важливості критеріїв та їхніх вагових коефіцієнтів є ключовим етапом. Після ранжування рішень за кожним критерієм вибирається підмножина Парето – рішення, які не мають інших, що б краще за них за будь-яким з критеріїв. Остаточний етап включає оптимізацію та аналіз вибраних рішень враховуючи конкретні умови задачі та вимоги. Цей процес дозволяє визначити оптимальні рішення та побудувати простір Парето – набір всіх можливих оптимальних рішень [7].

Отримані альтернативи за методом Парето наведено у таблиці 3.1. Обрання виконувалося за принципом відкидання найгіршої по усім параметрам альтернативи, тобто тієї альтернативи, що була гіршою за інші за усіма параметрами [7].

Таблиця 3.1 – Отримана підмножина альтернатив за методом Парето (таблиця виконана самостійно)

Архітектура/Шкала	Масштабованість	Гнучкість	Легкість в обслуговуванні	Продуктивність	Безпека
Мікро-сервісна	5	3	4	4	2

Кінець таблиці 3.1

Архітектура/Шкала	Масштабованість	Гнучкість	Легкість в обслуговуванні	Продуктивність	Безпека
Подієво-орієнтована	4	4	5	2	2
Клієнт-сервер	3	3	4	5	2
Мультирівнева	3	4	4	2	3
Сервісно-орієнтована	4	5	4	3	2

Отже, як можна побачити у таблиці 3.1, із множини альтернатив було видалено монолітний вид архітектури проектування та розробки веб програмної системи.

3.3 Нормування оцінок за шкалами з урахуванням min та max

Нормування оцінок за шкалами min та max – це метод, який використовується для стандартизації даних та усунення різниці в масштабах між оцінками [8]. Перший крок полягає у визначенні мінімального (min) та максимального (max) значень для конкретного параметра чи критерію. Це допомагає у створенні рамок, всередині яких будуть знаходитися значення параметрів для порівняння різних альтернатив. Далі розраховується різниця між кожним значенням та мінімальним значенням, і отримані значення нормуються у діапазоні від 0 до 1. Цей метод дозволяє забезпечити однаковий масштаб для оцінок з різних джерел, щоб зручно порівнювати та аналізувати дані [8]. Для нормування оцінок буде використано формулу нормування за міні-максом 3.1.

$$f = \frac{f_{\text{вимір}} - f_{min}}{f_{max} - f_{min}} \tag{3.1}$$

де $f_{\text{вимір}}$ – бал вимірювання;

f_{min} – мінімальний бал у цьому вимірюванні;

f_{max}– максимальний бал у цьому вимірюванні.

Результати розрахунків нормування оцінок наведено у таблиці оцінювання альтернатив за нормованими оцінками (див. табл. 3.2).

Таблиця 3.2 — Результати нормування оцінок альтернатив (таблиця виконана самостійно)

Архітектура/Шкала	Масштабованість	Гнучкість	Легкість в обслуговуванні	Продуктивність	Безпека
Мікро-сервісна	1	0.5	0.75	0.75	0.5
Подієво-орієнтована	0.75	0.75	1	0.25	0.5
Клієнт-сервер	0.5	0.5	0.75	1	0.5
Мультирівнева	0.5	0.75	0.75	0.25	1
Сервісно-орієнтована	0.75	1	0.75	0.5	0.5

Як можна побачити у таблиці 3.2, максимальна нормована оцінка, що могли отримати альтернативи дорівнює 1, а найменша оцінка, що отримали альтернативи дорівнює 0.5.

3.4 Визначення вагових коефіцієнтів простим ранжуванням

Визначення вагових коефіцієнтів простим ранжуванням є методом, що враховує важливість різних критеріїв чи факторів у прийнятті рішень. Перший крок цього процесу — визначення переліку критеріїв, які впливають на рішення. Далі проводиться просте ранжування цих критеріїв, зокрема, визначення їхньої важливості, без конкретного визначення числових вагових значень. Це може базуватися на експертних оцінках чи опитуваннях.

Після цього кожному критерію присвоюють ваговий коефіцієнт відповідно до його місця у ранжуванні. Важливою частиною є нормалізація вагових коефіцієнтів, забезпечуючи, що їхня сума дорівнює 1 або 100%. Це робить вагові коефіцієнти адаптованими до загального вагомого впливу на прийняття рішення.

Завершальний етап включає використання визначених вагових коефіцієнтів при оцінці альтернатив або виборі оптимального рішення. Важливо враховувати,

що цей метод ґрунтується на суб'єктивних експертних думках, тому результати можуть бути чутливими до індивідуальних оцінок та досвіду експертів.

Нехай ми визначимо важливість критерій наступним чином:

Безпека > продуктивність > масштабованість > гнучкість > легкість в обслуговуванні. Але в нас продуктивність залежить від масштабованості. Тоді вони матимуть наступну кількість балів:

– безпека = 5;
– продуктивність = 3.5;
– масштабованість = 3.5;
– гнучкість = 2;
– легкість в обслуговуванні = 1.

Наступним кроком необхідно розрахувати загальну кількість балів за формулою 3.2:

$$sum = \sum_{i=1}^{i=n} i, \tag{3.2}$$

де i – бали критерію;

sum – сума балів усіх критеріїв;

n – кількість критеріїв.

Використавши формулу 3.2, ми отримаємо значення параметру $sum = 15$. Вагові коефіцієнти розраховуються за наступною формулою:

$$c_i = \frac{i}{sum}, \tag{3.3}$$

де i – бали критерію;

sum – сума балів усіх критеріїв;

c_i – нормалізованій бали.

Отже, виконавши розрахунки за формулою 3.3, було отримано наступні вагові коефіцієнти:

– безпека = 0.33;
– продуктивність = 0.23;
– масштабованість = 0.23;
– гнучкість = 0.13;
– легкість в обслуговуванні = 0.066.

Дані показники необхідно використати для знаходження корисності за методом адитивної лінійної згортки з ваговими коефіцієнтами.

3.5 Розрахунок корисності альтернатив за методом адитивної згортки

Розрахунок корисності альтернатив за методом адитивної лінійної згортки є процесом, що дозволяє об'єднати значення критеріїв для оцінки альтернатив. Починається з вибору та визначення критеріїв, які важливі для прийняття рішення. Наступним кроком є нормалізація значень критеріїв для приведення їх до єдиного масштабу, включаючи в себе вагову оцінку та нормування оцінок за шкалами min та max [7].

Для розрахунку корисності було використано формулу 3.4 наведену нижче, яка дозволяє розрахувати корисність альтернативи із використанням вагових коефіцієнтів. Вагові коефіцієнти прийнято вказувати вже нормованими величинами.

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \tag{3.4}$$

де α_j – нормуючі показники;

n – кількість критеріїв;

β_j - вагові коефіцієнти, що відображають відносний внесок окремих критеріїв до загального критерію.

Далі наведено результати розрахунку корисності для кожної з альтернатив у підмножині Парето:

– мікро-сервісна: 0.682;

– подієво-орієнтована: 0.5585;
– клієнт-сервер: 0.6245;
– мультирівнева: 0.6495;
– сервісно-орієнтована: 0.632.

Дані корисності потрібно використати для ранжування альтернатив за корисністю. Вони дозволяють узагальнити різні аспекти альтернативи в єдиний числовий показник, який полегшує порівняння та вибір оптимального рішення.

3.6 Ранжування альтернатив за корисністю

Таким чином, результати отримані в пункті 3.5 дозволяють нам визначити корисність кожної альтернативи при розробці програмного веб-забезпечення, щоб ефективно ранжувати їх залежно від їхньої важливості для задоволення вимог та обмежень проектів. Це допомагає вибрати оптимальне рішення, яке найкраще відповідає потребам проекту та забезпечує успішну реалізацію програмного забезпечення.

Отже, було отримано наступне ранжування видів архітектури від найбільш корисної до найменш корисної:

– мікро-сервісна;
– мультирівнева;
– сервісно-орієнтована;
– клієнт-сервер;
– подієво-орієнтована;
– монолітна.

Дане ранжування надає нам можливість обрати правильний тип архітектури при розробці та проектуванні веб-системи на платформі .Net.

Ранжування альтернатив по вибору архітектури веб-системи дозволяє об'єктивно порівнювати альтернативи, враховуючи важливість конкретних критеріїв. Вагові коефіцієнти допомагають визначити вагомість кожного критерію, а ранжування створює пріоритетний порядок альтернатив. Цей процес мінімізує

суб'єктивність, забезпечуючи систематизований підхід до вибору архітектури та оптимізацію вибору за багатокритеріальними аспектами [7].

За допомогою даного ранжирування, є можливість зробити висновки, що найбільш ефективним є використання мікро-сервісної архітектури для розробки веб-системи на платформі .Net. Мікро-сервісна архітектура забезпечує високу гнучкість, масштабованість та незалежність окремих компонентів системи, що особливо важливо для сучасних веб-додатків з динамічним навантаженням та частими оновленнями [8].

Використання інших типів архітектури, крім мікро-сервісної, також можливе, проте їх ефективність може бути меншою при розробці веб-систем. Наприклад, монолітна архітектура може бути доцільною для невеликих проектів з обмеженим функціоналом та не вимагати високого рівня масштабованості. У таких випадках, коли потрібно швидко розвернути прототип чи невелику систему, монолітна архітектура може бути оптимальним варіантом, оскільки вона проста у використанні та розвитку.

Клієнт-серверна архітектура, з іншого боку, може бути вибрана у випадках, коли важливо забезпечити високу продуктивність взаємодії між клієнтською та серверною частинами системи. Це особливо актуально для веб-додатків, де значна частина обчислень відбувається на серверній стороні, наприклад, в ігрових або фінансових системах.

Отже, раціональний підхід до вибору архітектури веб-системи враховує специфічні потреби проекту, вагу критеріїв оцінки та забезпечує оптимальне рішення. Важливо об'єктивно оцінити всі можливі варіанти і вибрати той, який найбільше відповідає потребам проекту з урахуванням його масштабу, функціональних вимог та очікувань користувачів. Тільки такий підхід сприятиме ефективній реалізації програмного продукту та досягненню його поставлених цілей.

ВИСНОВКИ

Разом із розвитком методів розробки програмного забезпечення збільшилася і кількість можливих типів архітектури розробки. Вибір архітектури розробки веб-системи почав мати визначальне значення для всього її життєвого циклу. Вірна архітектура позитивно впливає на продуктивність, масштабованість, безпеку, підтримку та розширюваність системи. Вона також враховує швидкість розробки, вартість обслуговування, зручність для користувачів та технологічну актуальність. Правильний вибір архітектури сприяє успіху та готовності до майбутніх викликів, забезпечуючи ефективну роботу системи в довгостроковій перспективі [1].

В рамках цієї роботи було досліджено варіанти архітектури розробки та проектування веб-систем на платформі .Net. Було проаналізовано існуючі архітектурні рішення та обрано найбільш використовувані з них, знайдені їх недоліки та переваги.

На основі розглянутих питань та проведеного дослідження можна зробити наступні висновки: кожна система має свої певні переваги та недоліки, які розробники повинні враховувати при використанні певної методології. Найбільш ефективною для розробки веб-додатків було визначено мікро-сервісну архітектуру, яка базується на розподілі додатку на сервіси, які взаємодіють між собою. Найменш ефективною було визначено монолітну архітектуру, що ставить за основу розташування усіх елементів у одному кодовому монолітному базисі.

При виконанні дослідження, було проаналізовано різні способи вирішення задач багатокритеріального вибору та обрано метод адитивної лінійної згортки із ваговими коефіцієнтами, що дозволяє легко проаналізувати багато альтернатив, враховуючи різні критерії та важливість цих критеріїв [7].

Іншим важливим висновком є те, що різні архітектури розробки програмного забезпечення мають повністю протилежні недоліки та переваги. Тому не можна сказати, що, наприклад, монолітна архітектура є повністю неефективною. Вона може бути ефективною при розробці програмного забезпечення іншого типу, аніж веб-застосунків.

Отже, у результаті дослідження було виявлено, що мікро-сервісна архітектура дозволяє найкраще за інші отримати безпечну, легко масштабовану, дешеву у розробці, просту у підтримці веб-систему при використанні платформи .Net. Але при використанні іншої важливості критеріїв оцінювання, інший тип архітектури програмного забезпечення може мати перевагу над використанням мікро-сервісної архітектури, але це потребує іншого дослідження, використовуючи конкретні вимоги та пріоритети.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення / Роберт Мартін. – Харків: Фабула, 2019. – 416 с. – (1).

2. Волошин О. Ф. Моделі та методи прийняття рішень / О. Ф. Волошин, С. О. Мащенко. – Київ: Видавництво Людмила, 2018. – 292 с. – (3).

3. Ріхтер Д. CLR via C# / Джефрі Ріхтер. – Санкт-Петербург: Пітер Пресс, 2019. – 896 с. – (4).

4. Бібліотека MSDN [Електронний ресурс] - Режим доступу: www/ URL: http://msdn.microsoft.com/ru-ru/library/ee895049.aspx

5. Мартін Р. Чистий код. Створення і рефакторинг / Роберт Мартін. – Харків: Фабула, 2019. – 441 с. – (1).

6. Price Mark J. C# and .NET 6 / M. J. Price. – Boston: Packt Publishing, 2021. – 824 с. – (6).

7. Бондаренко М. Ф., Гвоздинський А. М. Оптимізаційні задачі в системах прийняття рішень: Підручник. - Харків: ХТУРЕ, 1998 - 216 с. ISBN 5-7763;

8. Наконечний О.Г. Методи прийняття рішень: навч. посіб. / О. Г. Наконечний, І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев; Мін-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків: ХНУРЕ, 2016. – 132 с.: іл. – ISBN 978-966-659-212-8