НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського
Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 Комп'ютерні науки
(код і назва)

на тему: Створення розподілених додатків на базі платформи .Net

Виконав (-ла): студент (-ка) 4 курсу, групи ДА-32
(шифр групи)

Чанцова Катерина Вікторівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к. т. н. Гіоргізова-Гай В.Ш. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Економічний доцент, к. е. н. Рощина Н.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль старший викладач Бритов О.А. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Чанцовій Катерині Вікторівні
(прізвище, ім'я, по батькові)

1. Тема роботи Створення розподілених додатків на базі платформи .NET, керівник роботи Гіоргізова-Гай В.Ш., доцент, к. т. н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 10 » травня 2017 р. №1477-с

2. Термін подання студентом роботи 15.06.2017

3. Вихідні дані до роботи
Платформа Microsoft.NET: WinForms, ASP.NET, ADO.NET;
мова програмування C#; середовище розробки Microsoft Visual Studio

4. Зміст роботи: Розподілені додатки, їх загальна структура.
Розповсюджені технології створення розподілених додатків, порівняльна характеристика.
Архітектура платформи .NET.
Реалізація лабораторного практикуму для створення розподілених додатків на платформі .NET
Функціонально-вартісний аналіз

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Загальна структура розподілених додатків (плакат)
Порівняльна характеристика платформи .NET та Enterprise JavaBeans (плакат)
Архітектура платформи .NET(плакат)
Лабораторний практикум (плакат)

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент, к. е. н		

7. Дата видачі завдання 17.01.17

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	17.01.17	
2	Збір інформації	17.04.17	
3	Аналіз можливостей платформ	07.05.17	
4	Складання завдань для лабораторного практикуму і розрахунково-графічної роботи	17.05.17	
5	Реалізація лабораторного практикуму	28.05.17	
6	Оформлення дипломної роботи	05.06.17	
7	Отримання допуску до захисту та подача роботи в ДЕК	15.06.17	
8	Захист дипломної роботи	19.06.17	

Студент _____ (підпис) _____ (ініціали, прізвище)

Керівник роботи _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

бакалаврської роботи Чанцової Катерини Вікторівни на тему:
«Створення розподілених додатків на базі платформи .Net»

Метою дипломної роботи є дослідження можливостей платформи .Net для створення розподілених додатків та створення лабораторного практикуму для ознайомлення з базовими можливостями платформи.

В роботі розглянуто основні властивості і вимоги до розподілених систем, приведено огляд основних видів їх архітектури, проаналізовано розповсюджені технології для створення розподілених додатків. Також в роботі проведено порівняльну характеристику найбільш поширених сьогодні платформ для створення розподілених додатків .Net та Enterprise JavaBeans. Для платформи .Net більш детально розглянуто її основні складові та зазначено можливості їх використання.

В практичній частині роботи реалізовано лабораторний практикум, метою якого є надання базових навиків з використання технології .Net для створення розподілених додатків.

Загальний обсяг роботи: 88 сторінок, 19 рисунків, 9 таблиць та 16 посилань.

Ключові слова: розподілені додатки, розподілені системи, платформа .Net, лабораторний практикум, архітектура системи, платформа Enterprise Java Beans.

АННОТАЦИЯ

бакалаврской работы Чанцовой Катерины Викторовны на тему:
«Создание распределенных приложений на базе платформы .Net»

Целью дипломной работы является исследование возможностей платформы .Net для создания распределенных приложений и создание лабораторного практикума для ознакомления с базовыми возможностями платформы.

В работе рассмотрено основные свойства и требования к распределенным системам, проведено обзор основных видов их архитектуры, проанализировано распространенные технологии для создания распределенных приложений. Также в работе проведено сравнительную характеристику наиболее распространенных сегодня платформ для создания распределенных приложений .Net и Enterprise JavaBeans. Для платформы .Net более детально рассмотрено её основные составляющие и приведены возможности их использования.

В практической части работы реализовано лабораторный практикум, целью которого является предоставление базовых навыков по использованию технологии .Net для создания распределенных приложений.

Общий объем работы: 88 страниц, 19 рисунков, 9 таблиц и 16 ссылок.

Ключевые слова: распределенные приложения, распределенные системи, платформа .Net, лабораторный практикум, архитектура системы, платформа Enterprise Java Beans.

ABSTRACT

for the bachelor's thesis of Chantsova Kateryna Viktorivna

"Developing of distributed applications based on the .Net platform"

This thesis is devoted to study the possibilities of the .Net platform for creation of distributed applications and to create a laboratory workshop to familiarize with the basic capabilities of the platform.

In this paper, the basic properties and requirements for distributed systems are considered, an overview of main types of architecture is given, analyzed common technologies for creating distributed applications. Also, the comparison of the most common contemporary platforms for creating distributed applications - .Net and Enterprise JavaBeans – was held. The main components of .Net platform are examined in more detail and the possibilities of using them are presented.

Practical part of the work implements a laboratory workshop, which purpose is to provide basic skills for using .Net technology to create distributed applications.

The thesis contains 88 pages, 19 figures, 9 tables and 16 references.

Key words: distributed applications, distributed systems, .Net platform, laboratory workshop, system architecture, Enterprise JavaBeans platform.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

SOA – Service Oriented Architecture.

SOAP – Simple Object Access Protocol, протокол обміну структурованими повідомленнями у розподіленому обчислювальному середовищі.

WSDL – Web Services Description Language, мова опису веб-інтерфейсів та доступу до них, заснована на мові XML.

RPC – Remote Procedure Call, віддалений виклик процедур.

DCOM – Distributed COM, розширення Component Object Model для підтримки зв’язку між об’єктами на різних комп’ютерах мережі.

CORBA – Common Object Request Broker Architecture.

HTTP – HyperText Transfer Protocol.

XML – eXtensible Markup Language.

UDDI – Universal Description Discovery & Integration, інструмент для розміщення описів веб-сервісів для подальшого їх пошуку іншими організаціями та інтеграції в свої системи.

JEE – Java Platform, Enterprise Edition.

P2P – peer-to-peer, рівний до рівного.

EJB – Enterprise Java Beans.

ORB – Object Request Broker, програмне забезпечення, що забезпечує зв’язок між об’єктами.

API – Application Program Interface.

CLR – Common Language Runtime, загальномовне середовище виконання.

JIT – Just-in-time

COM – Component Object Model, технологічний стандарт Microsoft, призначений для створення програмного забезпечення на основі взаємодіючих компонент.

VB – Visual Basic.

ASP – Active Server Pages, технологія розробки веб-додатків від Microsoft.

JSP - Java Server Pages, технологія розробки Java веб-додатків.

MSMQ – Microsoft Message Queuing, реалізація черги повідомлень від Microsoft.

ADSI – Active Directory Service Interfaces, інтерфейс програмування додатків, розроблений компанією Microsoft і призначений для доступу до різних служб каталогу, в першу чергу до Active Directory.

IIS – Internet Information Server, власний набір серверів для декількох служб Інтернету від компанії Microsoft.

RMI – Remote Method Invocation, програмний інтерфейс виклику віддалених методів мови Java.

WPF – Windows Presentation Foundation, система для побудови клієнтських додатків, створення інтерфейсу користувача, складова платформи .NET.

WCF – Windows Communication Foundation, служба для обміну даними між компонентами додатку на базі платформи .NET.

ВСТУП

Багато сьогоднішніх задач в науці, техніці, промисловості та інших областях є масштабними та багатоплановими. Для їх реалізації необхідно поєднати знання, вміння, засоби, можливості, програмне забезпечення та дані багатьох організацій, які часто знаходяться у віддалених куточках країни, та навіть світу. Два десятиліття тому дуже важко було забезпечити сумісну роботу різних організацій, що використовують в своїй роботі різні операційні системи, бази даних та протоколи зв'язку. Зростання потреби у співпраці між віддаленими організаціями призвело до розвитку систем і технологій, що поєднують поодинокі комп'ютери і дозволяють їм працювати як одне ціле. Такі системи отримали назву розподілених.

Питання створення розподілених систем не втратило актуальності. З появою нових технологій створення таких систем та вдосконаленням існуючих, змінювались та зростали і вимоги, що висуваються до розподілених систем. Основними вимогами були і залишаються масштабованість системи, підтримка логічної цілісності даних, стійкість системи та безпечність.

Можливість найбільш повного дотримання усіх вимог до розподіленої системи залежить не тільки від технології, за допомогою якої розроблена система, в більшому степені забезпечення вимог залежить від архітектури самої системи. Основні види архітектури розподілених систем: клієнт-серверна, сервіс-орієнтована, децентралізована. Кожен з видів має свої переваги та область застосування, та на різних рівнях забезпечує головні вимоги до розподілених систем.

В даній роботі розглянуто поняття розподіленої системи, її архітектури, надається опис основних видів архітектури, принципи їх побудови та особливості використання. Також досліджуються розповсюджені технології створення розподілених систем, проводиться їх порівняльна характеристика.

Основною метою роботи є створення на основі розглянутих матеріалів лабораторного практикуму. Лабораторний практикум має надавати базові знання

та навики роботи з основними засобами платформи .NET, призначеними для розробки розподілених додатків.

1 РОЗПОДІЛЕНІ СИСТЕМИ

За твердженням Е. Таненбаума розподілену систему можна визначити, як набір з'єднаних каналами зв'язку незалежних комп'ютерів, які з точки зору користувача деякого програмного забезпечення виглядають єдиним цілим.

У розподілених системах функції одного рівня додатку можуть бути рознесені між декількома комп'ютерами. З іншого боку, програмне забезпечення, встановлене на одному комп'ютері, може відповідати за виконання функцій, що відносяться до різних рівнів. Тому підхід до визначення розподіленої системи, що визначає її сукупністю комп'ютерів, умовний. Для опису та реалізації розподілених систем було введено поняття програмної компоненти.

Програмна компонента - це одиниця програмного забезпечення, що виконується на одному комп'ютері в межах одного процесу, і яка надає певний набір сервісів, які використовуються через її зовнішній інтерфейс іншими компонентами, як виконуються на цьому ж комп'ютері, так і на віддалених комп'ютерах. Ряд компонент користувацького інтерфейсу надають свій сервіс кінцевому користувачеві.

Ґрунтуючись на визначенні програмної компоненти, можна дати більш точне визначення розподіленої системи. Згідно з ним, розподілена система є набір взаємодіючих програмних компонент, що виконуються на одному або декількох пов'язаних комп'ютерах і виглядають з точки зору користувача системи як єдине ціле.

Прозорість є атрибутом розподіленої системи. При справному функціонуванні системи від кінцевого користувача має бути приховане, де і як виконуються його запити.

Програмна компонента є мінімальною одиницею розгортання розподіленої системи. В ході модернізації системи одні компоненти можуть бути оновлені незалежно від інших компонент [1].

1.1 Архітектура розподілених систем

Архітектура – базова організація системи, втілена у її компонентах, їх відносинах між собою та оточенням, а також принципи, що визначають проектування та розвиток системи. Для кожної розподіленої системи архітектура є дуже важливою, оскільки охоплює не тільки її структурні аспекти, а і правила її використання та інтеграції з іншими системами, функціональність, продуктивність, гнучкість та надійність. Архітектура також впливає на те, яким буде інтерфейс користувача.

1.1.1 Архітектура клієнт-сервер

Архітектура клієнт-сервер - це базова модель організації розподілених систем. У цій моделі всі процеси діляться на дві групи. Процеси, що реалізують певну роботу, наприклад, службу файлової системи або бази даних, називаються серверами. Процеси, що запитують служби у серверів шляхом посилки запиту і подальшого очікування відповіді від сервера, називаються клієнтами. Нерідко клієнти і сервери взаємодіють через комп'ютерну мережу і можуть бути як різними фізичними пристроями, так і програмним забезпеченням [2].

Найчастіше додатки типу клієнт-сервер поділяють на три логічних (рис. 1): призначений для користувача інтерфейс (ІК), логіка додатку (ЛД) і доступ до даних (ДД), що працює з базою даних (БД). Користувач системи взаємодіє з нею через інтерфейс користувача, база даних зберігає дані, що описують предметну область додатка, а рівень логіки додатка реалізує всі алгоритми, які стосуються предметної області.

Рисунок 1 - Логічні рівні додатку [1]

Дволанкова архітектура клієнт-сервер

Одним з варіантів клієнт-серверної архітектури є класична дволанкова архітектура (Two-tier architecture). Під клієнт-серверних додатком в цьому випадку розуміється інформаційна система, заснована на використанні серверів баз даних.

Схематично таку архітектуру можна представити, як показано на рис. 2.

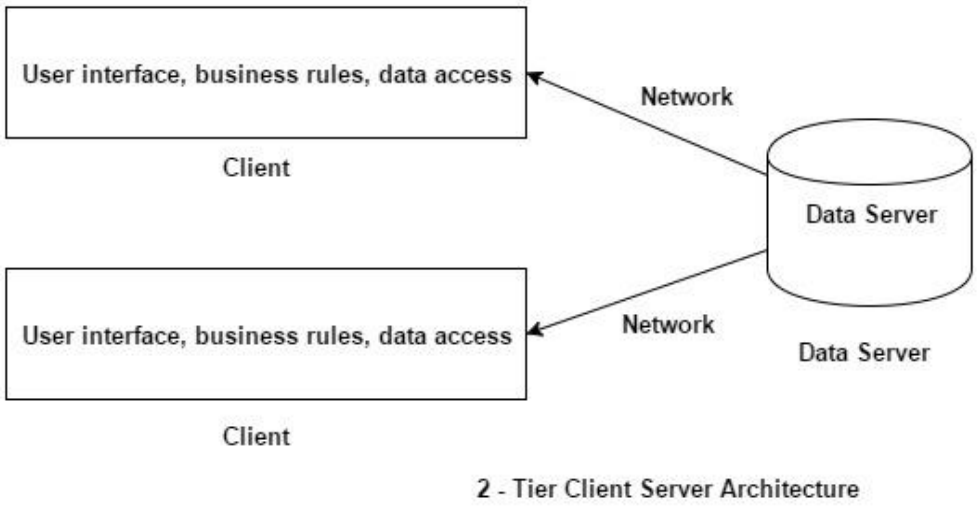

Рисунок 2 - Класичне уявлення архітектури клієнт-сервер

На стороні клієнта виконується код додатка, в який обов'язково входять компоненти, що підтримують інтерфейс з кінцевим користувачем, створюють звіти, виконують інші специфічні для додатку функції. Також клієнтська частина програми взаємодіє з клієнтською частиною програмного забезпечення управління базами даних.

На практиці системи, побудовані на дволанковій архітектурі, часто не відносять до класу розподілених, але формально вони можуть вважатися найпростішими представниками розподілених систем [1].

Перевагами даної архітектури є:

• можливість, в більшості випадків, розподілити функції обчислювальної системи між декількома незалежними комп'ютерами в мережі;

• всі дані зберігаються на сервері, який, як правило, захищений набагато краще за більшість клієнтів, а також на сервері простіше забезпечити контроль повноважень, щоб дозволити доступ до даних тільки клієнтам з відповідними правами доступу;

• підтримка роботи з багатьма користувачами;

• гарантія цілісності даних.

Недоліки:

• непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу;

• висока вартість обладнання;

• бізнес логіка додатків залишається в клієнтському програмному забезпеченні.

Триланкова архітектура клієнт-сервер

Найбільш поширеною є триланкова архітектура (three-tier, у якої інтерфейс користувача, логіка програми та доступ до даних виділені в самостійні складові системи, які можуть працювати на незалежних комп'ютерах (рис. 3).

Рисунок 3 - Триланкова архітектура [1]

Запит користувача в подібних системах послідовно оброблюється клієнтською частиною системи, сервером логіки додатка і сервером баз даних (рис. 4).

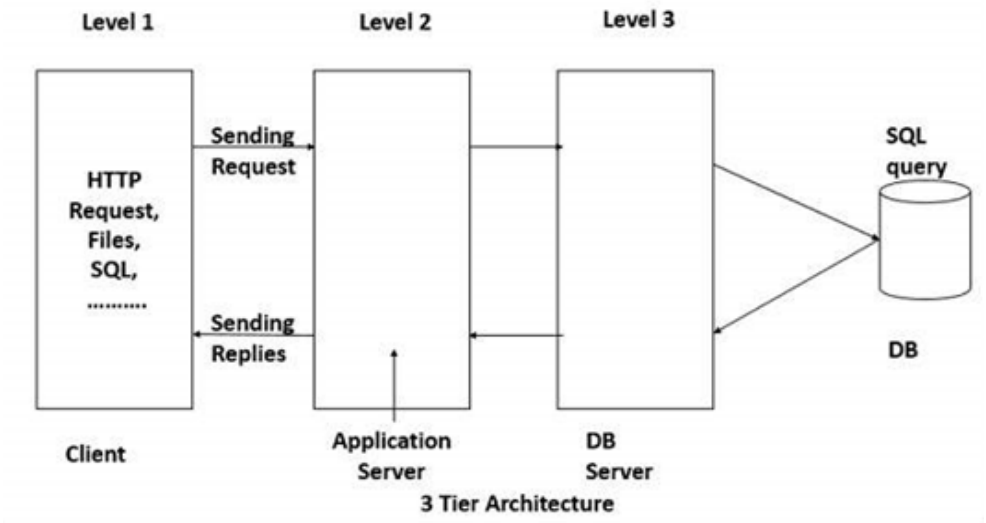

Рисунок 4 - Обробка запиту користувача в триланковій архітектурі

Дана архітектура має такі переваги:

• клієнтське програмне забезпечення не потребує адміністрування;

• масштабованість;

• конфігурованість - ізольованість рівнів один від одного дозволяє швидко і простими засобами переконфігурувати систему при виникненні збоїв або при плановому обслуговуванні на одному з рівнів;

• висока безпека та надійність;

• низькі вимоги до швидкості каналу (мережі) між терміналами і сервером додатків;

• низькі вимоги до продуктивності і технічних характеристик терміналів, як наслідок зниження їх вартості.

Недоліки:

• зростає складність серверної частини і, як наслідок, витрати на адміністрування і обслуговування;

• більш висока складність створення додатків;

• більша складність в розгортанні і адмініструванні;

• високі вимоги до продуктивності серверів додатків і сервера бази даних, і, як наслідок, і висока вартість серверного обладнання;

• високі вимоги до швидкості каналу (мережі) між сервером бази даних і серверами додатків [2].

Багатоланкова архітектура клієнт-сервер

Багатоланкова архітектура клієнт-сервер - це пряме продовження поділу додатків на рівні інтерфейсу користувача, логіки додатку та доступу до даних.

Можливі різні варіанти реалізації багатоланкової архітектури. Один з них: різні ланки взаємодіють відповідно до логічної організації додатку. Такий тип розподілу називається вертикальним. Головна його особливість - це розміщення логічно різних компонентів на різних машинах.

Інший вид розподілу - горизонтальний розподіл. При такому типі розподілу клієнт або сервер може містити фізично розділені частини логічно однорідного модуля, причому робота з кожною з частин може відбуватися незалежно. Це робиться для вирівнювання навантаження.

Є й інші варіанти організації архітектури, наприклад, розподілена як вертикально, так і горизонтально [3].

1.1.2 Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA, service-oriented architecture) - модульний підхід до розробки програмного забезпечення, заснований на використанні сервісів (служб) зі стандартизованими інтерфейсами (рис. 5).

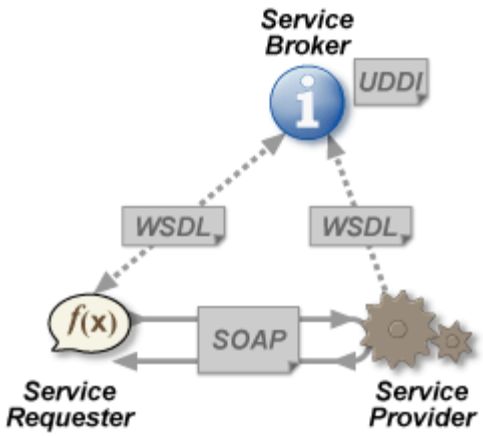

Рисунок 5 - Сервіс-орієнтована архітектура [16]

Сервіс (служба) - це компонента програми, безпосередньо доступна користувачу.

OASIS (Організація з розповсюдження відкритих стандартів структурованої інформації) визначає SOA наступним чином (OASIS Reference Model for Service Oriented Architecture V 1.0): сервіс-орієнтована архітектура - це парадигма організації та використання розподілених інформаційних ресурсів, таких як додатки і дані, що знаходяться в сфері відповідальності різних власників, для досягнення бажаних результатів споживачем, яким може бути кінцевий користувач або інша програма.

В основі SOA лежать принципи багатократного використання функціональних елементів, ліквідації дублювання функціональності в програмному забезпеченні, уніфікації типових операційних процесів, забезпечення переводу операційної моделі компанії на централізовані процеси і функціональну організацію на основі промислової платформи інтеграції.

Компоненти програми можуть бути розподілені по різних вузлах мережі, і пропонуються як незалежні, слабо пов'язані, замінні сервіси-додатки. Програмні комплекси, розроблені відповідно до SOA, часто реалізуються як набір веб-сервісів, інтегрованих за допомогою відомих стандартних протоколів (SOAP, WSDL, і т. п.)

Інтерфейс компонентів SOA-програми представляє собою інкапсуляцію деталей реалізації конкретного компонента (операційної системи, платформи, мови програмування, вендора,.) від інших компонентів. Таким чином, SOA надає гнучкий і елегантний спосіб комбінування і багаторазового використання компонентів для побудови складних розподілених програмних комплексів.

SOA добре зарекомендувала себе для побудови великих корпоративних програмних додатків. Цілий ряд розробників та інтеграторів пропонують інструменти і рішення на основі SOA (наприклад, платформи IBM WebSphere, Oracle / BEA Aqualogic, Microsoft Windows Communication Foundation, SAP NetWeaver, ІВК Юпітер, TIBCO, Diasoft).

Основними цілями застосування SOA для великих інформаційних систем, рівня підприємства, і вище є:

• скорочення витрат при розробці додатків, за рахунок упорядкування процесу розробки;

• розширення повторного використання коду;

• незалежність від використовуваних платформ, інструментів, мов розробки;

• підвищення масштабованості створюваних систем;

• поліпшення керованості створюваних систем.

Принципи SOA:

• архітектура не прив'язана до якоїсь певної технології;

• незалежність організації системи від використовуваної обчислювальної платформи (платформ);

• незалежність організації системи від вживаних мов програмування;

• використання сервісів, незалежних від конкретних додатків, з однаковими інтерфейсами доступу до них;

• організація сервісів як слабкозв’язаних компонентів для побудови систем.

Те, що архітектура не прив'язана до якоїсь певної технології, означає можливість її реалізації з використанням широкого спектру технологій, включаючи такі технології як REST, RPC, DCOM, CORBA або веб-сервіси. SOA може бути реалізована, використовуючи один з цих протоколів і, наприклад, може використовувати, додатково, механізм файлової системи, для обміну даними.

Web-сервіси базуються на широко поширених і відкритих протоколах: HTTP, XML, UDDI, WSDL і SOAP. Саме ці стандарти реалізують основні вимоги SOA - по-перше, сервіс повинен піддаватися динамічному виявленню і виклику (UDDI, WSDL і SOAP), по-друге, повинен використовуватися незалежний від платформи інтерфейс (XML). Нарешті, HTTP забезпечує функціональну HTTP сумісність. Web-сервіси розглядаються як ефективний інструмент для інтеграції, в тому числі для взаємодії процесів, які виконуються в різних компаніях. Особливе місце серед різних специфікацій, призначених для опису систем і додатків на рівні бізнес-процесів, займає мову BPEL4WS.

Головне, що відрізняє SOA, це використання незалежних сервісів, з чітко визначеними інтерфейсами, які, для виконання своїх завдань, можуть бути викликані якимось стандартним способом. Це здійснюється за умови, що сервіси заздалегідь нічого не знають про програму, яка їх викличе, а додаток не знає , яким чином сервіси виконують своє завдання.

SOA також може розглядатися як стиль архітектури інформаційних систем, який дозволяє створювати додатки, побудовані шляхом комбінації слабкозв’язаних і взаємодіючих сервісів. Ці сервіси взаємодіють на основі будь-якого чітко визначеного платформо-незалежного та мовно-незалежного інтерфейсу (наприклад, WSDL). Визначення інтерфейсу приховує мовно-залежну реалізацію сервісу.

Таким чином, системи, засновані на SOA, можуть бути незалежні від технологій розробки і платформ (таких як Java, .NET і т. д.). Наприклад, сервіси, написані на C #, що працюють на платформах .Net і сервіси на Java, що працюють на платформах Java EE, можуть бути з однаковим успіхом викликані загальним складеним додатком. Програми, що працюють на одних платформах, можуть викликати сервіси, що працюють на інших платформах, що полегшує повторне використання компонентів.

SOA може підтримувати інтеграцію і консолідацію операцій в складі складних систем, однак SOA не визначає і не надає методологій або фреймворків для документування сервісів [2].

1.1.3 Децентралізована архітектура

Однорангова, децентралізована або пірингова (від англ. Peer-to-peer, P2P - рівний до рівного) мережа - це комп'ютерна мережа, заснована на рівноправності учасників. У такій мережі відсутні виділені сервери, а кожен вузол (peer) є як клієнтом, так і сервером. На відміну від архітектури клієнт-сервер, така організація дозволяє зберігати працездатність мережі при будь-якій кількості і будь-якому поєднанні доступних вузлів.

Кожен комп’ютер при децентралізованій архітектурі є незалежним від інших, містить тільки ту інформацію, з якої повинен працювати, а актуальність даних у всій системі забезпечується завдяки безперервному обміну повідомленнями з іншими комп’ютерами. Обмін повідомленнями між ними може бути реалізований різними способами, від відправки даних по електронній пошті до передачі даних по мережах.

Однією з переваг такої схеми експлуатації та архітектури системи, є забезпечення можливості персональної відповідальності за збереження даних. Так як дані, доступні на конкретному робочому місці, знаходяться тільки на цьому комп'ютері, при використанні засобів шифрування і особистих апаратних ключів виключається доступ до даних сторонніх осіб.

Прикладом системи, побудованої на децентралізованій архітектурі є Skype. Каталог користувачів Skype розподілений по комп'ютерам користувачів мережі Skype, що дозволяє мережі легко масштабуватися до дуже великих розмірів без дорогої інфраструктури централізованих серверів.

Єдиним центральним елементом для Skype є сервер ідентифікації, на якому зберігаються облікові записи користувачів і резервні копії їх списків контактів. Центральний сервер потрібен тільки для встановлення зв'язку. Після того як зв'язок встановлено, комп'ютери пересилають голосові дані безпосередньо один одному (якщо між ними є прямий зв'язок) або через Skype-посередник (супервузол - комп'ютер, у якого є зовнішня IP-адреса і відкритий TCP-порт для Skype). Зокрема, якщо два комп'ютери, що знаходяться всередині однієї локальної мережі, встановили між собою Skype-з'єднання, то зв'язок з Інтернетом можна перервати і розмова буде тривати аж до її завершення користувачами або будь-якого збою зв'язку всередині локальної мережі [3].

1.2 Вимоги до розподіленої системи

Щоб досягти мети свого існування - поліпшення виконання запитів користувача - розподілена система повинна задовольняти деяким необхідним вимогам. Можна сформулювати наступний набір вимог, яким в найкращому випадку повинна задовольняти розподілена обчислювальна система.

Відкритість. Всі протоколи взаємодії компонент всередині розподіленої системи в ідеальному випадку повинні бути засновані на загальнодоступних стандартах. Це дозволяє використовувати для створення компонент різні засоби розробки і операційні системи. Кожна компонента повинна мати точну і повну специфікацію своїх сервісів. В цьому випадку компоненти розподіленої системи можуть бути створені незалежними розробниками. При порушенні цієї вимоги може зникнути можливість створення розподіленої системи, що охоплює кілька незалежних організацій.

Масштабованість. Масштабованість обчислювальних систем має кілька аспектів. Найбільш важливий з них - можливість додавання в розподілену систему нових комп'ютерів для збільшення продуктивності системи, що пов'язано з поняттям балансування навантаження (load balancing) на сервери системи. До масштабування відносяться також питання ефективного розподілу ресурсів сервера, який обслуговує запити клієнтів.

Підтримка логічної цілісності даних. Запит користувача в розподіленій системі повинен або коректно виконуватися цілком, або не виконуватися взагалі. Ситуація, коли частина компонент системи коректно обробили запит, що надійшов, а частина - ні, є найгіршою.

Стійкість. Під стійкістю розуміється можливість дублювання декількома комп'ютерами одних і тих самих функцій або можливість автоматичного розподілу функцій усередині системи у разі виходу з ладу одного з комп'ютерів. В ідеальному випадку це означає повну відсутність унікальної точки збою, тобто вихід з ладу одного будь-якого комп'ютера не приводить до неможливості обслужити запит користувача.

Безпека. Кожен компонент, який утворює розподілену систему, повинен бути впевнений, що його функції використовуються авторизованими на це компонентами або користувачами. Дані, що передаються між компонентами, повинні бути захищені як від спотворення, так і від перегляду третіми сторонами.

Ефективність. У вузькому сенсі стосовно до розподілених систем під ефективністю буде розумітися мінімізація накладних витрат, пов'язаних з розподіленим характером системи. Оскільки ефективність в даному вузькому сенсі може суперечити вимогам безпеки, відкритості та надійності системи, то слід зазначити, що ця вимога в даному контексті є найменш пріоритетною. Наприклад, на підтримку логічної цілісності даних в розподіленої системі можуть витрачатися значні ресурси часу і пам'яті, проте система з недостовірними даними навряд чи потрібна користувачам. Бажаною властивістю проміжного середовища є можливість організації ефективного обміну даними, якщо взаємодіючі програмні компоненти знаходяться на одному комп'ютері.

Ефективне проміжне середовище повинне мати можливість організації їх взаємодії без порушення стека TCP / IP. Для цього можуть використовуватися системні сокети (unix sockets) в POSIX системах або іменовані канали (named pipes).

Стійкість розподіленої системи пов'язана з поняттям масштабованості, але не еквівалентна йому. Припустимо, система використовує набір оброблюючих запити серверів і один диспетчер запитів, який розподіляє запити користувачів між серверами. Така система може вважатися досить добре масштабованою, проте диспетчер є вразливою точкою такої системи. З іншого боку, система з єдиним сервером може бути стійка, якщо існує механізм його автоматичної заміни у випадку виходу його з ладу, проте вона навряд чи відноситься до класу добре масштабованих систем. На практиці досить часто зустрічаються розподілені системи, що не відповідають даним вимогам: наприклад, будь-яка система з унікальним сервером бази даних, реалізованим у вигляді єдиного комп'ютера, має унікальну точку збою. Виконання вимог стійкості і масштабованості зазвичай пов'язане з деякими додатковими витратами, що на практиці може бути не завжди доцільно. Однак використовувані при побудові розподілених систем технології повинні допускати принципову можливість створення стійких і високо масштабованих систем [4].

Класичним прикладом системи, що значною мірою відповідає всім представленим вище вимогам, є система перетворення символьних імен в мережеві IP-адреси (DNS).

1.3 Висновки до розділу

В даному розділі було надане визначення розподілених систем та перераховані основні вимоги, що до них висуваються. Також було наведено визначення поняття архітектури розподіленої системи, описані такі можливі варіанти архітектури: архітектура клієнт-сервер (дволанкова, триланкова та багатоланкова), сервіс-орієнтована архітектура, децентралізована архітектура.

Найбільш поширеною архітектурою розподілених систем є клієнт-серверна триланкова архітектура. Вона вважається класичною, оскільки інші види архітектури (окрім децентралізованої) є її продовженням. Три головні рівні системи з триланковою архітектурою: інтерфейс користувача, логіка програми та доступ до даних, є самостійними складовими і можуть працювати на різних комп’ютерах. Такий розподіл за рівнями здається найбільш логічним і зрозумілим. Серед головних переваг даної архітектури гарна масштабованість системи, висока безпека та надійність, відсутність необхідності адміністрування клієнтського програмного забезпечення.

2 ТЕХНОЛОГІЇ СТВОРЕННЯ РОЗПОДІЛЕНИХ ДОДАТКІВ

До розповсюджених технологій створення розподілених додатків можна віднести такі технології: CORBA(Common Object Request Architecture), EJB(Enterprise Java Beans) і .NET.

2.1 CORBA

CORBA (Common Object Request Broker Architecture) – це набір відкритих специфікацій інтерфейсів, що визначає архітектуру технології межпроцессного і платформо-незалежного маніпулювання об'єктами. Розробниками даних інтерфейсів є OMG і X / Open. Object Management Group, Inc. (OMG) - це міжнародна організація, заснована в 1989 р, що складається більш ніж з 800 членів: постачальників інформаційних систем, розробників програмного забезпечення і користувачів. OMG просуває теорію і практику об'єктно-орієнтованій технології в область практичної розробки програмного забезпечення. Цей процес включає в себе розробку промислових стандартів і специфікацій управління об'єктами з метою створення загальної бази для розробки програмного забезпечення. Першочерговими завданнями є: повторне використання, мобільність і інтероперабельність (в термінології CORBA - можливість побудови системи одночасно на різних платформах і різних мережевих протоколах) об'єктно-орієнтованого програмного забезпечення в розподілених, гетерогенних середовищах. Підтримка даних стандартів створює можливість розробляти гетерогенні додатки, що працюють на всіх основних платформах і операційних системах. X / Open - незалежна всесвітня відкрита організація, яку підтримувала більшість найбільших постачальників інформаційних систем, призначених для користувацьких організацій і компаній-виробників програмного забезпечення. X / Open розробляє на основі існуючих та тих стандартів, що створюються, всеосяжне і інтегроване системне середовище

- Common Applications Environment (CAE). Компоненти CAE визначені в стандартах X / Open CAE. Основна мета CAE - створення пакетів програмних інтерфейсів (API) які можуть застосовуватися на практиці зі збереженням максимальної переносимості на рівні вихідних кодів програм.

Концептуальною інфраструктурою, на якій базуються всі специфікації OMG, є Object Management Architecture (OMA). До складу OMA входять різноманітні стандартизовані або ті, що стандартизуються зараз OMG, роботи, послуги, програмні зразки і шаблони (CORBAservices, horizontal and vertical CORBAfacilities), мова визначення інтерфейсів розподілених об'єктів IDL (Interface Definition Language), стандартизовані або ті, що стандартизуються відображення IDL на мови програмування і, нарешті, об'єктна модель CORBA.

Архітектура CORBA

CORBA визначає, яким чином програмні компоненти, розподілені по мережі, можуть взаємодіяти один з одним незалежно від оточуючих їх операційних систем і мов реалізації. Центральним елементом архітектури CORBA є ORB (Object Request Broker) - програмне забезпечення, що забезпечує зв'язок між об'єкти, в тому числі дозволяє знайти віддалений об'єкт за об'єктним посиланням (IOR - Interoperable Object Reference), викликати метод віддаленого об'єкта, передавши йому вхідні параметри (marshaling parameters), отримати значення, що повертаються, і вихідні параметри (unmarshaling parameters). ORB є сполучною ланкою між розподіленими частинами заснованої на технології CORBA системи, дозволяючи одній частині системи не зважати на фізичне розташування інших частин (об'єктів) системи. На ринку представлені ORB різних виробників, але всі вони відповідають єдиній специфікації CORBA. Тому CORBA дозволяє будувати розподілені системи, одночасно використовуючи ORB різних виробників, і будуючи систему одночасно на різних платформах і різних мережевих протоколах.

Також CORBA включає в себе наступні частини:

- Object Services - об'єктні сервіси, реалізації об'єктів, що надають загальні для об'єктно-орієнтованого середовища можливості:

служба імен, служба подій, служба збереження в довгостроковій пам'яті, служба транзакцій і так далі.

- Common Facilities - спільні засоби, це реалізації об'єктів, необхідні для великого числа додатків, наприклад, підтримка складених документів, потоків завдань.
- Application и Domain Interfaces - прикладні та галузеві інтерфейси. Прикладні об'єкти являють собою реалізації об'єктів для конкретних користувацьких додатків. В CORBA є також поняття домену. Реалізації об'єктів домену (CORBA domains) призначені для додатків з вертикальною організацією.

В CORBA інтерфейс об'єкта задається за допомогою мови опису інтерфейсів (Interface Definition Language, IDL). Тип об'єкта - це тип його інтерфейсу. Інтерфейс ідентифікується ім'ям, представленим ланцюжком символів. У моделі CORBA визначено базовий тип для всіх об'єктів - CORBA:: Object. Об'єкт підтримує тип свого безпосереднього інтерфейсу і за принципом успадкування всі його базові типи.

CORBA IDL задає визначення, які можуть відображатися в безлічі різних мов, не вимагаючи при цьому жодних змін від цільової мови. Ці відображення реалізуються компілятором IDL.

CORBA вводить поняття об'єктного посилання (object reference), яке унікальним чином ідентифікує об'єкт в мережі.

Механізм довгострокового зберігання, тобто збереження стану об'єкта в довготривалій пам'яті для подальшої його реактивації, в CORBA абсолютно прозорий для клієнта.

Найбільш поширені служби CORBA - служби іменування, управління життєвим циклом і подіями. Також CORBA підтримує інтерфейс динамічного виклику DII (Dynamic Invocation Interface).

В CORBA з самого початку була закладена багатоплатформенність і підтримка безлічі популярних мов програмування без необхідності будь-яких змін в них [5].

2.2 EJB

Enterprise JavaBeans (також часто вживається у вигляді абревіатури EJB) - специфікація технології написання і підтримки серверних компонентів, що містять бізнес-логіку. Є частиною Java EE. Додатки, створені за допомогою EJB, є масштабованими, орієнтованими на транзакції і безпечними при роботі в режимі багатьох користувачів. Ці додатки, одного разу написані, можуть потім бути розгорнуті на будь-який серверній платформі, що підтримує специфікацію EJB.

Enterprise Java Beans - це стандартна модель серверних компонентів для моніторів компонентних транзакцій. EJB-компоненти є Java (JEE) об'єктами, що реалізують технологію Enterprise Java Beans (EJB). Кожен такий компонент виконується під управлінням сервера додатків, який повинен відповідати так званій специфікації EJB- контейнера, тобто підтримувати відповідний API - EJB Container API (зазвичай сервер додатків в такому випадку називають EJB-контейнером). EJB-контейнер надає компонентам (Enterprise Beans) сервіси системного рівня (наприклад, багатопоточність, механізм транзакцій), залишаючись при цьому прозорим для розробника додатків. Ці системні сервіси дозволяють розробнику швидко створювати і розгортати EJB-компоненти: можна сказати, що контейнер «закриває» від розробника EJB всі складнощі системного характеру (наприклад, вже згадані багатопоточність або механізм транзакцій), дозволяючи йому зосередитися виключно на бізнес-логіці програми.

EJB-компонент являє собою Java-клас, який реалізує деяку бізнес-логіку. Всі інші класи в EJB-системі або реалізують підтримку клієнт-серверних взаємодій між компонентами, або реалізують деякі сервіси для компонентів.

Компонент EJB визначається як комбінація трьох складових елементів і опису його втановлення і застосування:

- home-інтерфейс, home-об’єкт,
- remote-інтерфейс, об’єкт EJB - реалізація remote-интерфейсу (EJBObject),
- Безпосередньо реалізація Enterprise Bean - це код реалізації бізнес-логіки.

- Опис встановлення EJB і його застосування. EJB-контейнер реалізує для компонентів, які знаходяться в ньому, такі сервіси, як транзакції (transaction), управління ресурсами, управління версіями компонентів, їх мобільністю, налаштовуваністю, життєвим циклом. Розробник EJB-компонента може просто викликати відповідні методи у контейнера. Клієнтські програми викликають методи на віддалених EJB-компонентах через EJB-об'єкт (EJB-object). EJB-об'єкт реалізує "віддалений інтерфейс" (remote interface) EJB-компонента на сервері. EJB-об'єкт реалізує лише бізнес-інтерфейс для EJB-компонента, будучи, в деякому сенсі, "проміжною" ланкою між клієнтом і EJB-компонентом.

EJB-об'єкти і EJB-компоненти являють собою різні класи. Хоча вони реалізують один і той же інтерфейс (інтерфейс, описаний для EJB-компонента), але при цьому вони виконують абсолютно різні функції. EJB-компонент виконується на сервері, усередині EJB-контейнера і реалізує бізнес-логіку, в той час як EJB-об'єкт виконується на стороні клієнта і віддалено викликає методи EJB-компонента.

Існує кілька причин, за якими використання Enterprise Bean-компонентів спрощує розробку великих розподілених корпоративних додатків.

Першою причиною є той факт, що EJB-контейнер надає всі необхідні сервіси системного рівня, дозволяючи розробнику компонента сконцентруватися на вирішенні бізнес-завдань. Саме EJB- контейнер (а не розробник) є відповідальним за забезпечення працездатності таких механізмів, як транзакції і авторизація доступу.

Другою перевагою EJB-компонентів є їх розташування на сервері. Внаслідок цього, розробникам клієнтів не доводиться включати бізнес-логіку до складу клієнтської програми - в такому коді не повинно бути функціональності, що реалізує бізнес-правила або доступ до баз даних. В результаті, клієнтське додаток виходить набагато меншого розміру, що дуже важливо для виконання на пристроях з обмеженими ресурсами.

Третьою перевагою Enterprise Java Bean-компонент є їх переносимість, збирач додатків може збирати нові додатки з уже існуючих компонент. Такі додатки можуть бути запущені на будь-якому JEE-сумісному сервері [5].

2.3 .NET

Під платформою Microsoft.NET слід розуміти інтегровану систему (інфраструктуру) засобів розробки, розгортання і виконання складних, розподілених програмних систем.

Тут набір базових класів забезпечує роботу з рядками, введення-виведення даних, багатопоточність і багато іншого. Набір класів для роботи з даними надають можливість використання SQL-запитів, ADO.Net і обробки XML даних.

Загальномовне середовище виконання (Common Language Runtime, CLR) активізує виконуваний код, виконує для нього перевірку безпеки, розміщує цей код в пам'яті і виконує його, забезпечує збірку сміття. Для забезпечення можливості багатомовної розробки ПО програмний код, одержуваний після компіляції програми на одній з алгоритмічних мов платформи MS.Net, представляється загальною проміжною мовою (Common Intermediate Language або CIL). Збірки (файли на CIL) перед своїм виконанням за допомогою JIT-компілятора (Just-In-Time compilers) транслюютьься з програмного коду на проміжній мові (CIL-коду) в машинний (native) код платформи виконання [4].

Детальніше можливості платформи Microsoft.Net будуть розглянуті в наступному розділі.

2.4 Порівняння технологій EJB та .NET

Microsoft .NET - самостійне повноцінне середовище розробки, яке серед багатьох інших можливостей включає і розробку розподілених додатків. EJB - технологія створена з метою спрощення розробки розподілених додатків, є частиною Java EE. EJB, частина JEE, і Microsoft .NET надають набір стандартизованих модульних компонентів і сервісів. Маючи стандартні

компоненти і сервіси в своєму розпорядженні, розробники можуть сконцентруватися на побудові бізнес-логіки додатка, замість того щоб програмувати більш фундаментальні речі. Використання візуальних інструментів значно спрощує розробку додатків.

Ключове відмінність між Java і .NET полягає в тому, що JEE є відкритим стандартом, який працює на декількох платформах, в той час як .NET є власністю Microsoft і працює виключно на Windows. Продукти JEE доступні від багатьох постачальникыв, в той час як технологія .NET доступна тільки від Microsoft.

Ще одна відмінність між JEE і .NET полягає в тому, що, використовуючи JEE, а відповідно і EJB, можна писати тільки на одній мові - Java. У той час як .NET підтримує кілька мов - основними з яких є Microsoft Visual Basic, C ++ і C #. Інші мови також можуть підтримуватися .NET, якщо вони переписані для запуску в середовищі .NET [6].

JEE і .NET надають бібліотеки компонентів. Бібліотека компонентів JEE включає API ядра Java, який включає Enterprise JavaBeans. .NET надає бібліотеки класів, які містять керовані компоненти .NET. .NET також використовує попереднє покоління COM + компонентів в операційній системі Windows. У моделі JEE окремі операції кожного рівня виконуються в окремих «контейнерах». Контейнер - це частина середовища виконання, яка послідовно виконує завдання - наприклад, вилучення даних і створення веб-сторінки. У .NET немає контейнерів, але є кілька шляхів, за якими операції можуть бути написані або оброблені. JEE і .NET мають набір компонентів, сервісів і функцій, які забезпечують стандартний спосіб виконання таких завдань, як доступ до баз даних, створення сценаріїв веб-сторінок, обробка повідомлень і підключення до віддалених ресурсів [7]. Порівняння ключових функцій і сервісів JEE та .NET Наведено в таблиці нижче (табл. 1).

Таблиця 1 - Порівняння ключових функцій і сервісів JEE та .NET

Service or feature	**Microsoft .NET**	**JEE**
Language	C#, VB.NET, C++.NET, more languages	Java
Operating System	Windows	Multiple
Runtime	CLR	JVM
Server Components	.NET, COM+	EJBs
Client/GUI Components	.NET class	JavaBeans
Web Server Scripting	ASP.NET	JSP/Servlet
Data Access	ADO.NET	JDBC
Persistent Objects	Business Entity Comp.	EJB Entity Beans
Message Queueing	Sys. Messaging, MSMQ	JMS on (MQSeries)
Asynchronous Invocation	COM+ QC	EJB Message Beans
Remoting	SOAP, HTTP, DCOM	RMI-over-IIOP
Naming	ADSI	JNDI
XML	System XML	JAXP
HTTP Engine	IIS	Application Servers from multiple vendors
Web Services Support	Built-in	Add-on

Як вже говорилося раніше, EJB не залежить від платформи, це є його перевагою в порівнянні з платформою .NET. Однак, незабаром це може зміниться, так як Microsoft працює над можливістю розробки за допомогою .NET на інших платформах.

У додатках .NET безпекою і пам'яттю управляє CLR, в додатках, побудованих за допомогою EJB, ресурсами, пам'яттю і безпекою управляє контейнер.

Технологія EJB добре інтегрована з технологією CORBA, що в деяких випадках є перевагою перед технологією .NET. Додатки, що використовують

технологію .NET можуть також використовувати технологію CORBA, але вона не так добре інтегрована в .NET, як в EJB.

Компонент EJB оформлюється як невеликий набір класів і інтерфейсів на Java, а також має дескриптор розгортання (опис в певному форматі на основі XML конфігурації компонента в рамках контейнера). .NET - компоненти являють собою набір класів і конфігураційних файлів, що грають роль дескрипторів розгортання і також представлених в деякому форматі на основі XML [8].

Для подальшого порівняння технологій .NET і EJB будуть розглянуті деякі особливості роботи з ними, пов'язані з такими параметрами, як зв'язок, іменування, процеси і синхронізація, цілісність, відмовостійкість, захист, робота з XML.

Зв'язок між компонентами, які працюють в різних процесах і на різних машинах, забезпечується в EJB, в основному, двома способами: синхронний зв'язок - за допомогою реалізації віддаленого виклику методів на Java (Java RMI), асинхронний - за допомогою служби повідомлень Java (Java message service , JMS). Зв'язок між компонентами в рамках .NET здійснюється за допомогою механізму Remoting, що реалізує як RMI, так і асинхронну передачу повідомлень, більш новою і досконалою технологією є WCF (Windows Communication Foundation).

Пошук ресурсів за іменами та ідентифікаторами і наборами їх властивостей в рамках EJB здійснюється за допомогою інтерфейсу JND (Java Naming and Directory Interface). В .NET в разі, якщо фізичне положення компонентів, з якими необхідно встановити зв'язок, відомо і є постійним, використовується локальна служба іменування, вбудована в середу. В іншому випадку, використовується Active Directory. Оскільки ця технологія з'явилася раніше, ніж середовище .NET, в рамках .NET була створена бібліотека адаптерів, що дозволяють використовувати функції Active Directory.

Розбиття програми на набір взаємодіючих процесів і потоків та керування ними здійснюється EJB-контейнером автоматично. Крім процесів і потоків,

середа .NET підтримує так звані зони додатків (application domains). Вони, як і процеси, слугують агрегатами ресурсів, але управляються більш ефективними механізмами.

В рамках одного процесу може бути створено декілька зон додатків. Вони слугують додатковим елементом захисту .NET-додатків від ненавмисного впливу і дозволяють зберегти працездатність процесу у разі виникнення проблем в одному з його додатків. Окрім автоматично створюваних потоків і зон додатків, розробник може створювати свої власні потоки та зони додатків. Питання синхронізації потоків і передачі даних між зонами додатків можуть вирішуватися за допомогою стандартних механізмів .NET.

Цілісність і несуперечливість даних при роботі з обома платформами підтримується за допомогою механізму розподілених транзакцій.

Відмовостійкість додатків в .NET, як і в EJB, повинна забезпечуватися або за рахунок використання додаткових продуктів, або за рахунок специфічного проектування програми.

Захищеність додатків для обох платформ підтримується такими методами: техніки аутентифікації, можливість визначення ролей, забезпечених набором прав доступу до різних елементів системи, а також можливість використання різних протоколів шифрування і захищеної передачі даних, управління ключами і підтвердження цілісності даних.

У EJB робота з XML здійснюється за допомогою спеціальної бібліотеки, що стосується .NET, то можливість роботи з XML-даними є вбудованої в рамках механізмів ADO.NET [9].

Обидві технології, EJN та .NET, можуть використовуватися для розробки розподілених компонентів, але їх також можна використовувати локально. Це, однак, більш вірно для компонентів .NET, ніж для компонентів EJB, оскільки компоненти EJB оброблюються контейнером, клієнт якого підключається віддалено, якщо тільки клієнт не є іншим компонентом або сервлетом, розміщеним на одному і тому ж контейнері або сервері додатків відповідно.

.NET має сильні інструменти для інтерфейсу користувача і краще розвинену клієнтську сторону. В той же час Java має перевагу на боці сервера, оскільки сам по собі JEE являє собою набір специфікацій, сфокусованих на полегшення розробки серверних додатків, зокрема на Unix-системах. Набір інструментів .NET, в загальному випадку, вважається більш легким в застосуванні для розробки простих веб-сервісів, в той час як EJB розглядається як кращий варіант для розробки складних гетерогенних додатків [9].

Однією з сильних сторін EJB є те, що широкий вибір інструментів, продуктів і додатків доступний від багатьох постачальників. Конкуренція серед них сприяє постійному вдосконаленню технології. Розвиток же .NET повністю залежить від Microsoft, однак ця технологія молодша на декілька років, і при її створенні були враховані деякі недоліки вже існуючих технологій.

2.5 Висновки до розділу

Технологія CORBA довгий час була стандартом для створення розподілених систем. Вона стала основою для інших технологій, включаючи технології EJB та Microsoft.NET, але зараз поступово стає менш використовуваною.

Технології EJB і Microsoft .NET аналогічні за своєю природою, кожна з них має певні переваги і недоліки. Обидві технології в багатьох аспектах мають свій власний спосіб і тип реалізації для вирішення різних завдань, наприклад, для віддалених викликів. Крім цього, у них є підтримка багатьох подібних функцій.

Обираючи між двома технологіями, в першу чергу потрібно зважати на те, де буде використовуватись розроблювана система, які вимоги до неї висуваються, що є більш важливим: багатоплатформенність чи можливість вибору мови програмування, більш сильна клієнтська сторона чи серверна, участь у створенні системи розробників, які досконало знають одну з технологій, або вже наявне програмне забезпечення, що буде взаємодіяти з системою або використовуватись для розробки, також може стати одним з критеріїв вибору.

3 АРХІТЕКТУРА ПЛАТФОРМИ .NET

.NET Framework - програмна платформа, випущена компанією Microsoft в 2002 році. Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), яке підходить для різних мов програмування. Функціональні можливості CLR доступні в будь-яких мовах програмування, що використовують це середовище.

Основною ідеєю при розробці .NET Framework було забезпечення свободи розробника за рахунок надання йому можливості створювати додатки різних типів, здатні виконуватися на різних типах пристроїв і в різних середовищах. Другим принципом стала орієнтація на системи, що працюють під управлінням сімейства операційних систем Microsoft Windows.

Програма для .NET Framework, написана будь-якою мовою програмування, що підтримується, спочатку перекладається компілятором в єдиний для .NET проміжний байт-код Common Intermediate Language (CIL) (раніше називався Microsoft Intermediate Language, MSIL). У термінах .NET виходить збірка, англ. assembly. Потім код або виконується віртуальною машиною Common Language Runtime (CLR), або транслюється утилітою NGen.exe в виконуваний код для конкретного цільового процесора. Використання віртуальної машини є переважним, оскільки позбавляє розробників необхідності піклуватися про особливості апаратної частини. У разі використання віртуальної машини CLR вбудований в неї JIT-компілятор «на льоту» (just in time) перетворює проміжний байт-код в машинні коди потрібного процесора. Сучасна технологія динамічної компіляції дозволяє досягти високого рівня швидкодії. Віртуальна машина CLR також сама піклується про базову безпеку, управління пам'яттю та систему винятків, виконуючи частину роботи за розробника.

Архітектура .NET Framework описана і опублікована в специфікації Common Language Infrastructure (CLI), яка розроблена Microsoft та затверджена

ISO і ECMA. У CLI описані типи даних .NET, формат метаданих про структуру програми, система виконання байт-коду і багато іншого [10].

.NET є багаторівневим, модульним і ієрархічним. Кожен рівень .NET Framework є шаром абстракції. Мови .NET - це верхній і найбільш абстрагований від інших рівень. Common Language Runtime (CLR) - це нижній рівень, найменш абстрагований і найбільш близький до вихідного середовища. Це важливо, тому що CLR тісно пов'язана з операційним середовищем для управління додатками. .NET Framework розділений на модулі, кожен з яких має свою особливу сферу відповідальності. Оскільки вищі рівні запитують служби тільки з більш низьких рівнів, .NET є ієрархічним. Загальна архітектура .NET Framework показана на рис. 6.

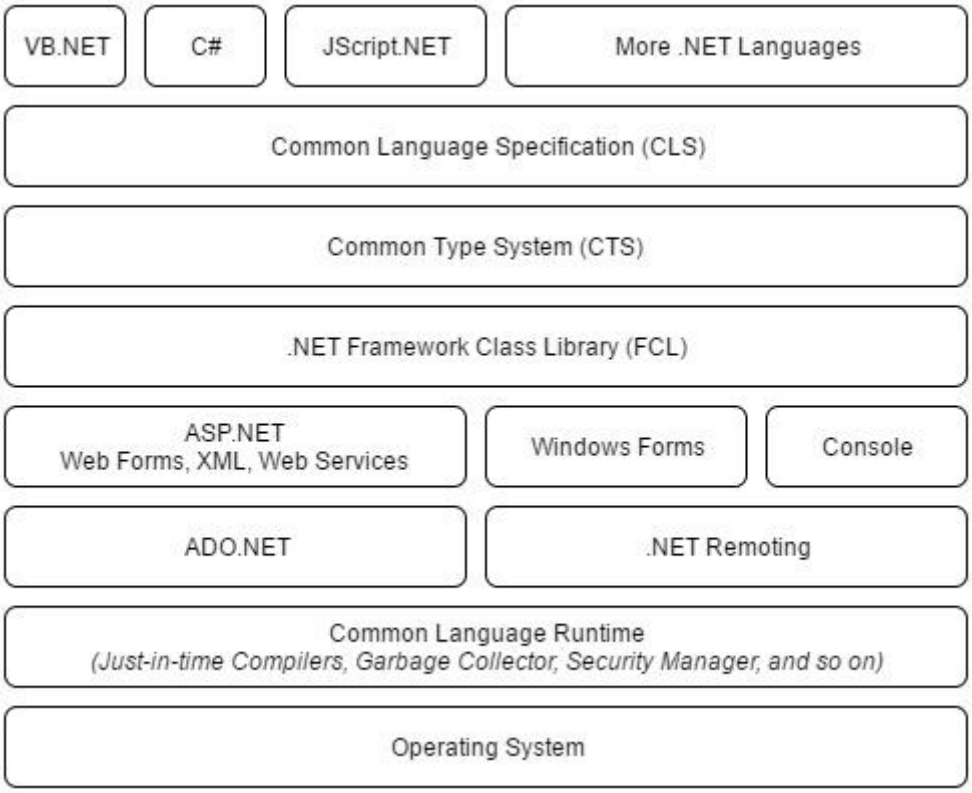

Рисунок 6 - Архітектура платформи .NET

Об'єктні класи .NET, доступні для всіх підтримуваних мов програмування, містяться в бібліотеці Framework Class Library (FCL). У FCL входять класи

Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation та інші (рис. 7). Ядро FCL називається Base Class Library (BCL).

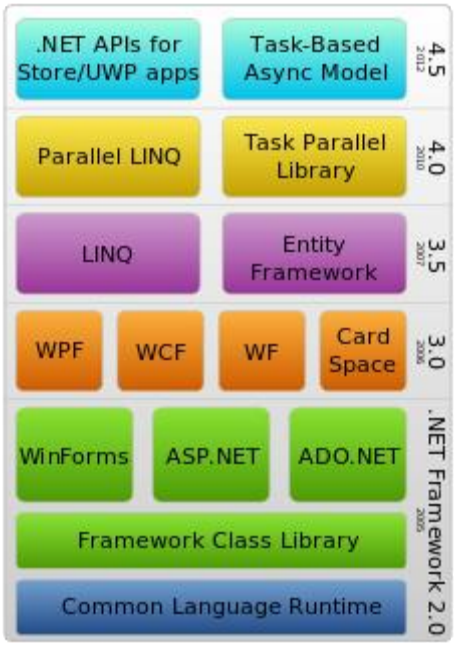

Рисунок 7 - Стек технологій .NET Framework [10]

3.1 Засоби створення інтерфейсу користувача

Платформа .NET Framework пропонує чотири варіанти API-інтерфейсу для застосування при побудові додатків з інтерфейсом користувача.

• ASP.NET (System.Web.UI). Призначений для створення додатків тонких клієнтів, які виконуються в стандартних веб-браузері.

• Silverlight. Призначений для побудови розширених користувацьких інтерфейсів всередині веб-браузера.

• Windows Presentation Foundation (System.Windows). Призначений для створення додатків товстих клієнтів.

• Windows Forms (System.Windows.Forms). Призначений для підтримки успадкованих додатків товстих клієнтів.

У загальному випадку додаток тонкого клієнта зводиться до веб-сайту; додаток товстого клієнта - це програма, яку кінцевий користувач повинен завантажувати або встановлювати на своєму комп'ютері. Успадковане програмне забезпечення – програмне забезпечення, що з деяких причин перестало задовольняти потребам, що змінилися, але продовжує використовуватись через перешкоди, що виникають при спробах його замінити.

3.1.1 ASP.NET

Програми, написані з використанням ASP.NET, розташовуються на сервері Windows IIS і можуть бути доступні за допомогою майже всіх веб-браузерів. Нижче перераховані переваги ASP.NET в порівнянні з технологіями товстих клієнтів.

• Відсутність необхідності розгортання на клієнтській стороні.

• Клієнти можуть використовувати платформи, відмінні від Windows.

• Просте розгортання оновлень.

Крім того, оскільки велика частина коду, який доводиться писати в додатку ASP.NET, виконується на сервері, рівень доступу до даних проектується для виконання в тому ж самому домені додатку - без обмеження безпеки або масштабованості. На противагу цьому, товстий клієнт, який робить те ж саме, в загальному випадку є не настільки безпечним та масштабованим. (Рішенням для товстого клієнта є створення проміжного рівня між клієнтом і базою даних. Цей проміжний рівень виконується на віддаленому сервері додатків (часто разом з сервером бази даних) і взаємодіє з товстими клієнтами через WCF, Web Services або Remoting.)

При написанні своїх веб-сторінок можна вибирати між традиційним API-інтерфейсом Web Forms і новим API-інтерфейсом MVC (Model-View-Controller - модель-уявлення-контролер). Обидва вони побудовані на основі технології ASP.NET. Технологія Web Forms була частиною .NET Framework з самого початку, а MVC реалізована набагато пізніше як реакція на успіх Ruby on Rails і MonoRail. Технологія MVC з'явилася в .NET Framework 4.0 і з того часу стала

більш досконалою. В цілому вона надає кращу програмну абстракцію, ніж Web Forms; також вона дозволяє краще контролювати HTML-розміткоу, що генерується. Однак є один аспект, в якому MVC програє Web Forms - візуальний конструктор. Тому застосування Web Forms є більш зручним для побудови веб-сторінок з переважно статичним вмістом.

Обмеження ASP.NET в значній мірі відображають загальні обмеження систем тонких клієнтів:

• інтерфейс веб-браузера істотно обмежує те, що можна робити;

• підтримка стану на стороні клієнта (або від імені клієнта) є громіздкою.

Проте, інтерактивність і чутливість створюваних за допомогою ASP.NET додатків можна поліпшити за допомогою сценаріїв клієнтської сторони або технологій на зразок AJAX. Робота з AJAX спрощується за рахунок використання таких бібліотек, як jQuery.

Типи, призначені для створення програмного забезпечення ASP.NET, знаходяться в просторі імен System.Web.UI і його підпросторах; вони упаковані в збірку System.Web.dll.

3.1.2 Silverlight

Формально Silverlight не є частиною .NET Framework: це окрема платформа, яка містить підмножину ключових засобів .NET Framework, а також підтримує можливість виконання у вигляді плагіна веб-браузера. Графічна модель Silverlight - по суті, підмножина WPF, це дозволяє застосовувати знання та навики роботи з останньою при розробці Silverlight-додатків. Плагін Silverlight доступний як міжплатформений завантажуваний файл для веб-браузерів, що дуже схоже на Macromedia Flash.

Flash володіє набагато більшою установною базою, тому домінує в цій галузі. З цієї причини Silverlight, як правило, використовується для написання сценаріїв, наприклад, в корпоративних мережах.

3.1.3 Windows Presentation Foundation (WPF)

Технологія WPF з'явилася в .NET Framework 3.0 і призначена для створення програмного забезпечення товстих клієнтів. Нижче перераховані переваги WPF в порівнянні з Windows Forms.

• Вона підтримує розвинену графіку, включаючи довільні трансформації, тривимірну візуалізацію і справжню прозорість.

• Первинна одиниця виміру заснована не на пікселях, тому додатки коректно відображаються за будь-якого налаштування DPI (dots per inch - точок на дюйм).

• Вона має велику підтримку динамічного компонування, що означає можливість локалізації додатку без небезпеки того, що елементи будуть перекривати один одного.

• Візуалізація використовує DirectX і є швидкою, отримуючи переваги від апаратного прискорення графіки.

• Користувацькі інтерфейси можуть бути описані декларативно в XAML-файлах, які підтримуються незалежно від файлів відокремленого коду - це допомагає відокремити зовнішній вигляд від функціональності [11].

Типи для написання WPF-додатків знаходяться в просторі імен System.Windows і всіх його підпросторах крім System.Windows.Forms.

3.1.4 Windows Forms

Windows Forms - це API-інтерфейс товстого клієнта, який є ровесником .NET Framework. У порівнянні з WPF це відносно проста технологія, яка пропонує більшість можливостей, необхідних при написанні типового Windows-додатку. Вона також важлива для підтримки успадкованих додатків. Однак якщо порівнювати з WPF, Windows Forms має низку недоліків.

• Позиції та розміри елементів управління задаються в пікселях, що призводить до ризику некоректного відображення додатків на клієнтах з налаштуваннями DPI, що відрізняються від налаштувань у розробників.

• API-інтерфейсом для малювання нестандартних елементів управління є GDI+, який, незважаючи на достатню гнучкість, повільно візуалізує великі області (і без подвійної буферизації може привести до мерехтіння).

• Важко домогтися надійності динамічного компонування.

Останній пункт є гарною причиною віддати перевагу WPF перед Windows Forms. Елементи компонування в WPF, подібні Grid, спрощують організацію міток і текстових полів таким чином, що вони будуть завжди вирівняними - навіть при зміні мови локалізації - без неакуратної логіки розташування і мерехтіння. Крім того, не доведеться приводити все до найменшого спільного знаменника в плані екранного дозволу - елементи компоновки WPF з самого початку проектувалися з підтримкою зміни розмірів.

Як позитивний момент слід відзначити, що технологія Windows Forms відносно проста у вивченні і як і раніше широко підтримується в елементах управління третіх сторін [12].

Типи Windows Forms знаходяться в просторах імен System.Windows.Forms (збірка System.Windows.Forms.dll) і System.Drawing (збірка System.Drawing.dll).

3.2 Засоби створення серверної частини

3.2.1 ADO.NET

ADO.NET - це керований API-інтерфейс доступу до даних. Назва технології породжена від назви застосовуваної в 1990-х роках технології ADO (ActiveX Data Objects - об'єкти даних ActiveX).

ADO.NET є набором класів, що реалізують програмні інтерфейси для полегшення підключення до баз даних із додатку незалежно від особливостей реалізації конкретної системи управління базами даних і від структури самої бази даних, а також незалежно від місця розташування цієї самої бази. Також ця технологія широко використовується спільно з технологією ASP.NET для доступу до розташованих на сервері баз даних з боку клієнта.

Основу інтерфейсу взаємодії з базами даних в ADO.NET представляє обмежене коло об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. За допомогою об'єкта Connection відбувається установка підключення до джерела даних. Об'єкт Command дозволяє виконувати операції з даними із бази даних. Об'єкт DataReader зчитує отримані в результаті запиту дані. Об'єкт DataSet призначений для зберігання даних з бази даних і дозволяє працювати з ними незалежно від неї. Об'єкт DataAdapter є посередником між DataSet і джерелом даних. Головним чином, через ці об'єкти відбувається робота з базою даних.

Однак щоб використовувати один і той же набір об'єктів для різних джерел даних, необхідний відповідний провайдер даних. Власне через провайдер даних в ADO.NET і здійснюється взаємодія з базою даних. Причому для кожного джерела даних в ADO.NET може бути свій провайдер, який і визначає конкретну реалізацію вищевказаних класів.

За замовчуванням в ADO.NET є наступні вбудовані провайдери:

• Провайдер для MS SQL Server

• Провайдер для OLE DB (Надає доступ до деяких старих версій MS SQL Server, а також до баз даних Access, DB2, MySQL і Oracle)

• Провайдер для ODBC (Провайдер для тих джерел даних, для яких немає своїх провайдерів)

• Провайдер для Oracle

• Провайдер EntityClient. Провайдер даних для технології ORM Entity Framework

• Провайдер для сервера SQL Server Compact 4.0

Крім цих провайдерів, які є вбудованими, існує також безліч інших, призначених для різних баз даних, наприклад, для MySQL.

Схематично архітектуру ADO.NET можна представити таким чином (рис.8):

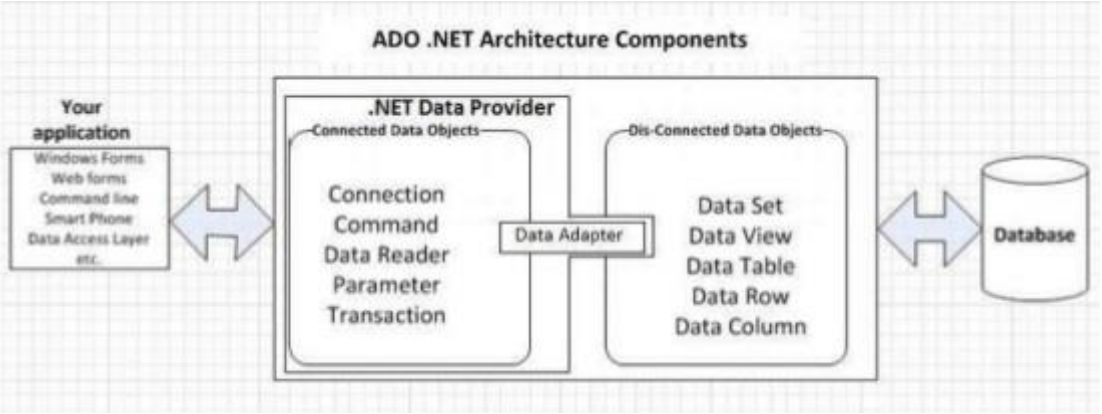

Рисунок 8 - Архітектура ADO.NET

Функціонально класи ADO.NET можна розбити на два рівня: підключений і відключений. Кожен провайдер даних .NET реалізує свої версії об'єктів Connection, Command, DataReader, DataAdapter і ряду інших, які складають підключений рівень. Тобто за допомогою них встановлюється підключення до бази даних і виконується взаємодія з нею. Як правило, реалізації цих об'єктів для кожного конкретного провайдера в своїй назві мають префікс, який вказує на провайдера.

Інші класи, такі як DataSet, DataTable, DataRow, DataColumn і ряд інших складають відключений рівень, так як після отримання даних в DataSet можливо працювати з цими даними незалежно від того, чи встановлено підключення до бази даних чи ні. Тобто після отримання даних із бази додаток може бути відключено від джерела даних [14].

3.2.2 Windows Workflow

Windows Workflow - це технологія для моделювання та управління бізнес-процесами, що виконуються потенційно довго. Робочий потік націлений на стандартну бібліотеку часу виконання, надаючи узгодженість і можливість взаємодії. Робочий потік також допомагає скоротити обсяг коду для динамічно керованих дерев прийняття рішень.

Windows Workflow не є строго серверною технологією - її можна використовувати де завгодно (наприклад, потік сторінки в інтерфейсі).

Концепція робочого потоку спочатку з'явилася у версії .NET Framework 3.0, з типами, визначеними в просторі імен System.WorkFlow. У .NET Framework 4.0 ця концепція була повністю переглянута; нові типи тепер знаходяться в просторі імен System.Activities.

3.2.3 COM + і MSMQ

Платформа .NET Framework дозволяє взаємодіяти з COM + для таких служб, як розподілені транзакції, через типи в просторі імен System. EnterpriseServices. Вона також підтримує MSMQ (Microsoft Message Queuing - організація черги повідомлень Microsoft) для асинхронного односпрямованого обміну повідомленнями за допомогою типів з простору імен System.Messaging.

3.3 Засоби створення розподілених систем

3.3.1 Windows Communication Foundation (WCF)

WCF являє собою складну технологію для комунікацій, яка з'явилася у версії .NET Framework 3.0. Вона є гнучкою і достатньо конфігурованою для того, щоб зробити зайвими своїх попередників - Remoting і Web Services (.ASMX).

WCF, Remoting і Web Services схожі між собою в тому, що всі вони реалізують описану нижче базову модель комунікацій між клієнтським і серверним додатками.

• На стороні сервера вказується, які методи можуть бути викликані віддаленими клієнтськими додатками.

• На стороні клієнта вказуються сигнатури серверних методів, які повинні викликатися.

• На сторонах сервера і клієнта вибираються транспортний і комунікаційний протокол (в WCF це робиться через прив'язку).

• Клієнт встановлює підключення до сервера.

• Клієнт викликає віддалений метод, який прозоро виконується на сервері.

WCF зменшує зв'язаність клієнта і сервера за допомогою контрактів служб і контрактів даних. Концептуально замість того, щоб безпосередньо викликати віддалений метод, клієнт відправляє повідомлення (XML або бінарне) кінцевій точці віддаленій служби. Одна з переваг такого методу полягає в тому, що клієнти не мають ніяких залежностей від платформи .NET або від будь-яких патентованих комунікаційних протоколів.

Технологія WCF є виключно конфігурованою і пропонує найбільш широку підтримку для стандартизованих протоколів обміну повідомленнями, включаючи WS- *. Це дозволяє взаємодіяти з учасниками, які виконують інше програмне забезпечення - можливо, на різних платформах, - і в той же час підтримувати розширені засоби на зразок шифрування. Ще одна перевага WCF полягає в тому, що існує можливість змінювати протоколи без необхідності зміни інших аспектів клієнтського або серверного додатка.

Типи, що мають відношення до WCF, знаходяться в просторі імен System.ServiceModel.

3.3.2 Remoting і .ASMX Web Services

Remoting і .ASMX Web Services - це попередники WCF, які з появою останньої стали менш використовуваними, хоча Remoting все ще використовується в комунікаціях між доменами додатків всередині одного і того ж процесу.

Функціональність Remoting орієнтована на додатки з сильною зв'язністю. Типовим прикладом може служити ситуація, коли і клієнт, і сервер є додатками .NET, написаними однією компанією (або компаніями, які поділяють спільні збірки). Комунікації зазвичай передбачають обмін потенційно складними спеціальними об'єктами .NET, які технологія Remoting серіалізує і десеріалізує без будь-якого втручання ззовні.

Функціональність Web Services орієнтована на додатки зі слабкою зв'язністю або додатки в стилі SOA. Типовим прикладом може служити ситуація, коли сервер спроектований на прийом простих SOAP-повідомлень, які

ініціюються клієнтами, що виконують різне програмне забезпечення, можливо, на різних платформах. Технологія Web Services може використовувати тільки HTTP і SOAP в якості транспортних і форматуючих протоколів, а додатки зазвичай розміщуються на сервері IIS. За переваги взаємодії доводиться платити зниженням продуктивності - додаток Web Services зазвичай повільніше, як при виконанні, так і при розробці, ніж добре спроектований додаток Remoting.

Типи для Remoting знаходяться в просторі імен System.Runtime.Remoting, а типи для Web Services - в просторі імен System.Web.Services.

3.3.3 CardSpace

CardSpace - це протокол аутентифікації на основі маркерів і управління ідентичністю, розроблений з метою спрощення управління паролями для кінцевих користувачів. Цією технології була приділена не дуже велика увага через труднощі в перенесенні маркерів між машинами (популярною альтернативою, позбавленої цієї проблеми, є OpenID).

Технологія CardSpace побудована на відкритих стандартах XML, тому в ній можуть брати участь сторони, які не залежать від Microsoft. Користувач може мати кілька сутностей, які управляються третьою стороною (постачальник сутностей). Коли користувач бажає отримати доступ до ресурсу на сайті X, він аутентифікується для постачальника сутностей, який потім видає маркер для сайту X. Це усуває необхідність надання пароля безпосередньо сайту X і скорочує кількість сутностей, якими користувач повинен керувати [11].

WCF дозволяє вказувати сутність CardSpace при підключенні через захищений канал HTTP за допомогою типів з просторів імен System.IdentityModel. Claims і System.IdentityModel.Policy.

3.4 Висновки до розділу

В рамках даного розділу були визначені основи архітектури та базові складові платформи .NET. Будь-яка конкретна складова платформи спрямована

на полегшення розробки певних фрагментів та логічних рівнів програмного забезпечення, найбільш повне задоволення потреб розробників. Як набір засобів розробки широко спектру застосування платформа .NET є потужним засобом для розробки складних систем.

Умовно всі засоби, які включає платформа .NET можна розділити на три групи: засоби створення користувацьких інтерфейсів, засоби створення серверної частини, засоби створення розподілених систем. В кожну з груп входять схожі за своїм призначенням технології, але їх відрізняють деякі особливості використання та область застосування.

Так поширеними засобами створення інтерфейсу користувача є ASP.NET, WPF та Windows Forms, робота з даними здійснюється переважно з використанням ADO.NET, а сучасним засобом організації взаємодії між компонентами розподіленої системи є служба WCF, яка прийшла на зміну .NET Remoting.

4 ЛАБОРАТОРНИЙ ПРАКТИКУМ

Для виконання лабораторних робіт необхідні встановлені на комп’ютері Microsoft Visual Studio (версія не нижче 2012) та Microsoft SQL Server (версія не нижче 2010) .

4.1 Лабораторна робота 1

Створення простої служби windows communication foundation (WCF)

Мета роботи: отримати базові знання та навики роботи з WCF.

Зміст роботи: визначення можливих шляхів створення служби WCF, робота служби WCF на прикладі простого додатку.

Інструкції до виконання роботи

Є декілька шляхів створення служби WCF, можна, наприклад, обрати в якості початкової точки стандартний шаблон проекту *Class Library* (Бібліотека класів) і вручну додати посилання на збірки WCF.

Але більш простим і зрозумілим є альтернативний шлях: обрати у Visual Studio шаблон проекту *WCF Service Library* (Бібліотека служб WCF). Цей тип проекту автоматично встановлює посилання на необхідні збірки WCF, однак його недоліком є те, що при цьому генерується деякий об’єм початкового коду, який згодом доволі часто видаляється.

Однією з переваг вибору цього шаблона є створення файлу *App.config*. Корисність цього файлу у тому, що при налагодженні або запуску проекту *WCF Service Library* інтегроване середовище розробки Visual Studio автоматично завантажить додаток *WCF Test Client* (Тестовий клієнт WCF). Програма *WcfTestClient.exe* читає налаштування із файлу *App.config,* тому може використовуватись для тестування служби.

У Visual Studio доступний ще один шаблон проекту *WCF Service*, пов'язаний з WCF, який знаходиться в діалоговому вікні *New Web Site*, що відкривається через пункт меню *File => New = > Web Site*. Даний шаблон

зручний у випадку, якщо заздалегідь відомо, що служба WCF має використовувати протоколи на основі HTTP. На противагу цьому, служба створена із застосуванням шаблона *WCF Service Library* дає можливість розміщення служби декількома способами.

Для створення додатку необхідно виконати наступні дії:

1. Створення служби WCF (рис. 9)

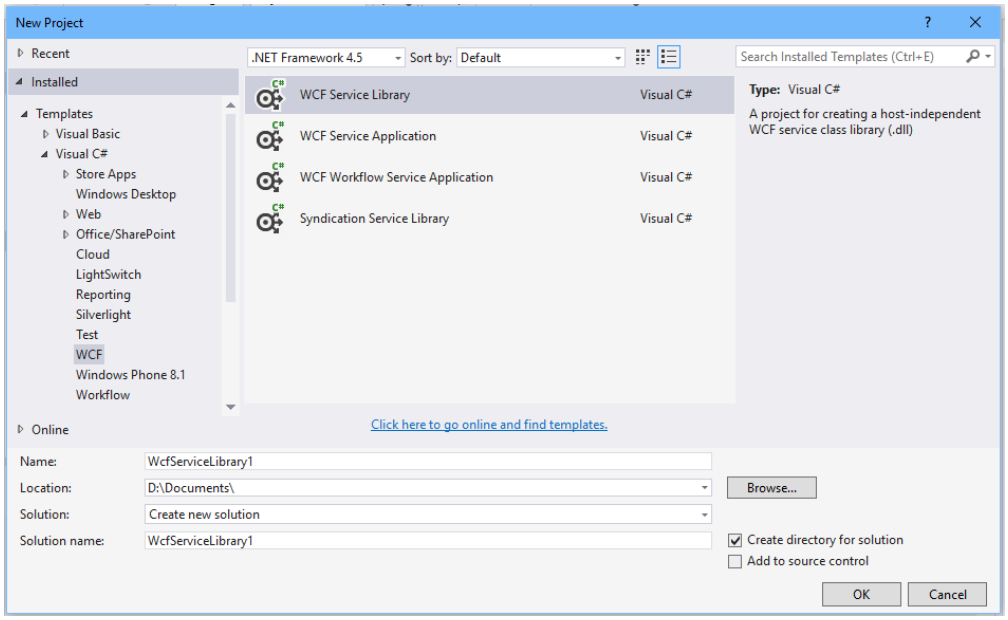

Рисунок 9 - Створення служби WCF

В діалоговому вікні обрати Visual C# => WCF => WCF Service Library. Вибрати директорію для збереження проекту та натиснути ОК. Таким чином буде створена працююча служба, яку можна буде тестувати та використовувати.

2. Тестування служби WCF

Натиснути F5 для запуску служби. На екрані з’явиться форма WCF Test Client та завантажить службу. Двічі натиснути метод GetData() під вузлом IService1. З’явиться вкладка GetData.

В області запиту ввести значення та натиснути кнопку Invoke (Виклик). Результат буде виведено в області відповіді.

3. Створення проекту сервісу (для прикладу взято калькулятор, який виконує чотири арифметичні дії)

   Створити проект типу *WCF Service Application* (розділ C#, WCF). Його структура буде такою:

   1) IService1.cs містить опис інтерфейсу сервісу, тобто набір методів, які сервіс надає.
   2) Service1.svc складається з двох частин - реалізація сервісу (Service1.svc.cs) і розмітка (Markup).
   3) Web.config - конфігурація сервісу.
   4) Папка App_Data

   Змінити назви файлів сервісу та його інтерфейсу, надавши їм імена Calculator та ICalculator відповідно. Перевірити наступний рядок у розмітці сервісу (файл Calculator.svc).

   ```
   <%@ ServiceHost Language="C#" Debug="true" Service="Project
   Name.Calculator" CodeBehind="Calculator.svc.cs" %>
   ```

   Оголосити стандартні арифметичні операції, такі як додавання (Addition), віднімання (Subtraction), множення (Multiplication) і ділення (Division).

   Додати метод TestConnection, який повертає рядок. За допомогою цього методу клієнти зможуть перевірити, що сервіс функціонує. Всі методи інтерфейсу повинні бути позначені атрибутом [OperationContract], інакше вони не будуть видимі клієнтам.

4. Розміщення сервісу

   Сам по собі сервіс являє собою бібліотеку та для його запуску у вигляді сервісу необхідно використовувати один із наданих WCF методів:

   - Хостинг на IIS
   - Запуск у вигляді служби Windows
   - Self hosting (сервіс виконаний у вигляді консольного додатку, що запускає сервіс)

Принципової різниці між цими методами немає. Найпростіший варіант – розміщення на IIS. Для цього потрібно відкрити конфігурацію проекту та обрати вкладку Web.

Розділ Start Action визначає, що відбудеться при запуску проекту:

- Current Page (за замовчуванням) – запуск останньої відкритої сторінки в браузері.
- Specific Page – запуск конкретної сторінки.
- Start External program – запуск вказаного виконуваного файла.
- Start URL – відкриття вказаного у полі URL адреси в браузері.
- Don’t open a page – нічого не відкривати, але чекати запита від стороннього додатку.

У всіх випадках сервіс завантажується в режимі налагодження та поводить себе як звичайний додаток Visual Studio. Останній варіант зручний тим, що нічого не відкривається у браузері.

Розділ Servers визначає, де буде розміщений сервіс. За замовчуванням обраний IIS Express (вбудований у Visual Studio варіант IIS),. Але можна розмістити сервіс і на звичайному IIS, для цього треба його встановити та увімкнути необхідні компоненти.

Вказати адресу сервісу в налаштуваннях та натиснути *Create Virtual Directory.*

Перевірити роботу додатку, відкривши в браузері рядок, вказаний у конфігурації проекту та додавши в кінці назву сервісу.

5. Публікація сервісу

Під публікацією сервісу мається на увазі його компіляція для подальшого викладення збірки на сервер.

Створити профіль публікації: натиснути правою кнопкою миші на імені проекту та обрати пункт Publish….

Майстер публікації запропонує декілька можливих варіантів.

Обрати Custom, ввести назву профіля. Вибрати метод публікації File System та вказати шлях, за яким буде збережена збірка, краще створити для неї нову папку.

Для розміщення сервісу на сайті, необхідно скопіювати папку з додатком на сервер та підключити до IIS як додаток, таким чином, якби це був звичайний сайт (Default Web Site => Add Application…).

6. Виклик сервісу

Використовувати створений додаток можна різними способами, далі будуть розглянуті наступні: виклик через клієнт на C#; звернення через SoapUI.

1) Клієнт на C#

Створити у наявному розв’язку ще один проект типу Console Application та назвати його TestClient. Додати посилання на сервіс (Add Service Reference).

У вікні, що відкрилося, ввести локальну адресу сервісу. Також в цьому вікні вказується назва простору імен та додаткові налаштування, видимі методи, що використовує створений сервіс.

Після створення посилання на сервіс, в проекті має з’явитися папка “Service References”, в якій буде знаходитись згенерований Visual Studio клієнт.

Написати код для підключення до сервісу та виклику арифметичних операцій, що ним надаються.

2) Звернення через SoapUI

SoapUI – безкоштовна програма для тестування SOAP-сервісів. Поміж іншого, вона надає можливість перегляду внутрішньої структури повідомлень, що може бути корисним при вирішенні складних проблем, у випадку коли клієнт і сервіс використовують різні платформи.

Завантажити програму та створити новий проект. В полі адреси сервісу вказати WSDL та ввести назву проекту.

Програма ввідобразить основні інтерфейси сервісу та створить тестові запити для кожного з методів.

Замінити простір імен в атрибуті сервісу ServiceContract на бажаний. Оновити конфігурацію сервісу в тестовому клієнті (RemoteService => Update Service Reference); оновити опис в SoapUI (Update Definition), натиснувши F5 на імені точки підключення BasicHttpBinding_ICalculator та оновити запит.

Завдання

1. Ознайомитися з інструкціями до виконання роботи.
2. Обрати завдання відповідно до варіанту (табл. 2).
3. Згідно до інструкцій написати програму, яка використовує службу WCF.
4. Перевірити роботу програми, зберегти скриншоти результатів.

Зміст звіту

1. Мета та зміст роботи.
2. Коротка характеристика служби WCF, її використання.
3. Результати роботи програми.
4. Висновки по роботі.

Контрольні запитання

1. Яке призначення служби WCF?
2. Яким чином відбувається взаємодія компонентів через службу WCF?
3. Чим WCF відрізняється від .Net Remoting?
4. Які переваги та недоліки використання WCF?

Таблиця 2 - Варіанти завдань до лабораторної роботи 1

Номер варіанту	Операції, які має виконувати калькулятор
1	Додавання, логарифмування, arccos, корінь
2	Віднімання, sin, arcctg, приведення до степеню
3	Множення, ctg, arcsin, експонента в степені
4	Ділення, cos, arctg, остача від ділення
5	Факторіал, tg, переведення градусів в радіани, цілочислове ділення
6	Переведення радіанів в градуси, tg, arccos, додавання
7	Обернене до числа, логарифмування, віднімання, факторіал
8	Остача віл ділення, факторіал, множення, корінь
9	Додавання, sin, ctg, переведення градусів в радіани
10	Віднімання, логарифмування, arcsin, cos
11	Множення, логарифмування, cos, експонента в степені
12	Ділення, переведення радіанів в градуси, логарифмування, факторіал
13	Додавання, arcctg, arcsin, факторіал
14	Віднімання, arccos, цілочислове ділення, arctg
15	Множення, переведення радіанів в градуси, arcctg, корінь
16	Ділення, факторіал, sin, arcsin
17	Додавання, приведення до степеню, остача від ділення, cos
18	Віднімання, корінь, експонента в степені, остача від ділення
19	Множення, факторіал, цілочислове ділення, приведення до степеню
20	Ділення, переведення градусів в радіани, ctg, cos

4.2 Лабораторна робота 2

Створення веб-додатку ASP.NET

Мета роботи: ознайомитися з можливостями ASP.NET, навчитися застосовувати дану технологію для створення веб-додатків.

Зміст роботи: створення простого додатку ASP.NET, робота з Entity Framework та базою даних.

Інструкції до виконання роботи

1. Створення нового ASP.NET проекту

   Створити новий проект Visual Studio. Обрати шаблон Asp.NET Web Application. Він знаходиться в Installed => Templates => Visual C# => Web. Ввести назву проекту та обрати його розміщення.
   В діалоговому вікні New ASP.NET Project обрати MVC.

2. Запуск додатку за замовчуванням

   Після завершення створення додатку у Visual Studio, запустити додаток, натиснувши Debug => Start Debugging або F5. Для ініціалізації Visual Studio та нового додатку може знадобитися час. Після завершення браузер відобразить запущений додаток.
   Для зупинки додатку закрити браузер та натиснути іконку “Stop Debugging” у Visual Studio.

3. Створення класів

   Додаток міститиме дві сутності:

   - Book
   - Author

   Вони будуть визначені у папці *Models* в Solution Explorer.
   Створити новий клас в папці *Models*. В полі Name написати “Author.cs” та натиснути OK.

Змінити автоматично згенерований код таким чином:

- підключити бібліотеки

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

- додати поля класу *AuthorID, LastName, FirstName, Books:*

```
[ScaffoldColumn(false)]
   public int AuthorID { get; set; }
   [Required]
   [Display(Name = "Last Name")]
   public string LastName { get; set; }

   [Display(Name = "First Name")]
   public string FirstMidName { get; set; }

   public virtual ICollection<Book> Books { get; set; }
```

Створити клас Book та змінити код, підключивши бібліотеку, як і минулому випадку, та додавши поля класу *BookID, Title, Year, Price, Genre, AuthorID*:

```
    [ScaffoldColumn(false)]
     public int BookID { get; set; }
     [Required]
     public string Title { get; set; }

     public int Year { get; set; }

     public double Price { get; set; }

     public string Genre { get; set; }

     [ScaffoldColumn(false)]
     public int AuthorID { get; set; }

     // Navigation property
   public virtual Author Author { get; set; }
```

Властивість *Author* визначає спосіб керування відносинами між автором і книгою (цей вид властивості називається *navigation property*).

Оскільки для зберігання моделей буде використана база даних SQLServer, то для роботи з об’єктами в базі даних потрібно визначити для них первинний ключ. Властивість моделі, яка буде ідентифікатором (первинним ключем), повинна мати таке ім’я: Ім’я_моделіID; інший спосіб визначити первинний ключ – додати потрібній властивості атрибут *Key*.

4. Встановлення в проект Entity Framework

   Для роботи з даними в ASP.NET MVC рекомендується використовувати Entity Framework, хоча це не є обов'язковим. Перевага його використання у тому, що він дозволяє абстрагуватися від структури певної бази даних і виконувати всі операції через модель.

   Для додання бібліотек Entity Framework в проект у вікні Solution Explore натиснути правою кнопкою миші на вузол References => Manage NuGet Packages. Знайти пакет Entity Framework та встановити його.

5. Додання даних

   В папці Models створити новий клас BookContext.

   Контекст даних використовує Entity Framework для доступу до бази даних на основі деякої моделі. Для створення контексту даних, новий клас має наслідувати клас DbContext. Такі властивості, як *Public DbSet<Book> Books { get; set; }* допомагають отримати з бази даних набір даних певного типу (наприклад, набір об'єктів Book).

   Відкрити файл web.config та знайти секцію *connectionStrings*. В цій секції визначається шлях до бази даних, яка буде створюватись.

   Змінити поле *name,* написавши ім'я класу контексту даних.

   Оскільки використовується Entity Framework таблиці бази даних будуть створені автоматично відповідно до моделей.

   Створити клас *BookDbInitializer* для додання початкових даних в базу даних. Для заповнення бази даних деякими початковими даними при кожному запуску додатку можна наслідувати цей клас від класу *DropCreateDatabaseAlways*. Для додання даних використати методи *db.Books.Add* та *db.Authors.Add.*

   Додати у файл *Global.asax* (в цьому файлі містяться початкові налаштування та конфігурації додатку) в метод *Application_Start* наступний рядок:

```
Database.SetInitializer(new BookDbInitializer());
```

6. Скаффолдинг

Скаффолдинг – метод програмування, для створення веб-додатків, що взаємодіють з базою даних. За підтримки цього методу середовищем, розробник не задає специфікацій, за якими в подальшому генерується програмний код для операцій створення визначених записів в БД, їх читання, оновлення та видалення (CRUD).

Натиснути правою клавішою миші на папку Controllers в Solution Explorer. Add => New Scaffolded Item. В діалоговому вікні вибрати MVC 5 Controller with views, using Entity Framework, натиснути кнопку Add.

Далі, в діалоговому вікні Add Controller в списку з модельними класами вибрати Book (Books.Models). Встановити контекстний клас на ApplicationDBContext (Book.Models). Прапорець Generate views має бути встановлений (рис. 10). Натиснути кнопку Add.

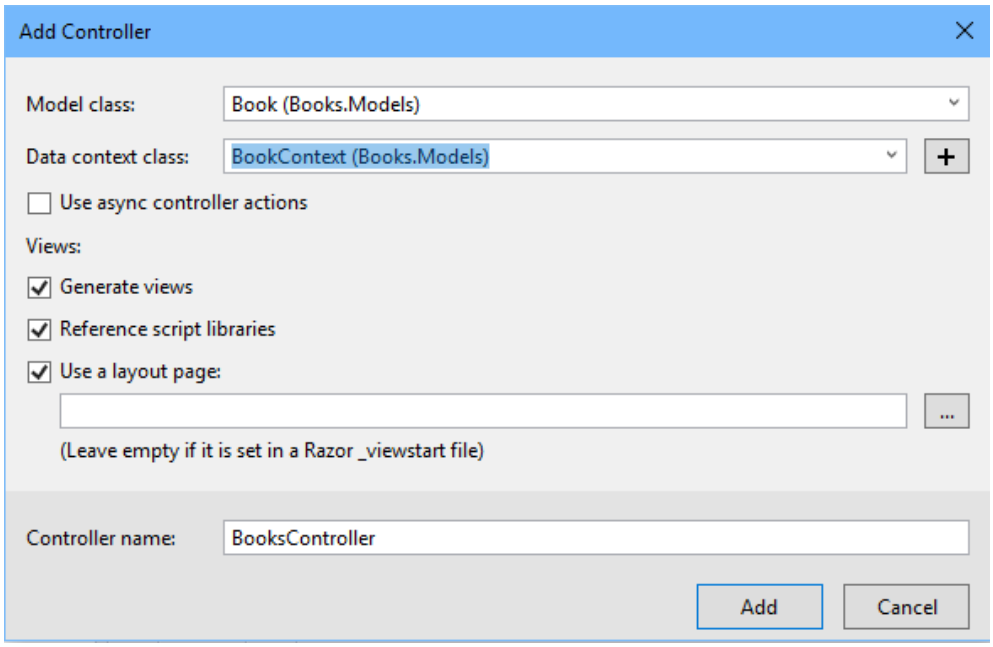

Рисунок 10 - Додання контроллеру

Результатом виконання перерахованих вище дій буде створення коду, який надає контроллер та набір представлень. В представленнях є UI та

код для створення, читання, оновлення, видалення та перелічення даних з бази даних.

За аналогією створити контроллер Author та пов'язані з ним представлення, для цього використати модельний клас Author (Book.Models) та контекстний клас ApplicationDBContext (Book.Models).

7. Створення додатку

Для відображення на веб-сторінці інформації, яка міститься у базі даних змінити відповідним чином файл Layout.cshtml.

Наприклад, це можна зробити додавши наступні рядки:

```
<li>@Html.ActionLink("Books", "Index", "Books")</li>
<li>@Html.ActionLink("Authors", "Index", "Authors")</li>
```

В меню Build вибрати Build Solution.

Після запуску додатку можна побачити наступне (рис. 11):

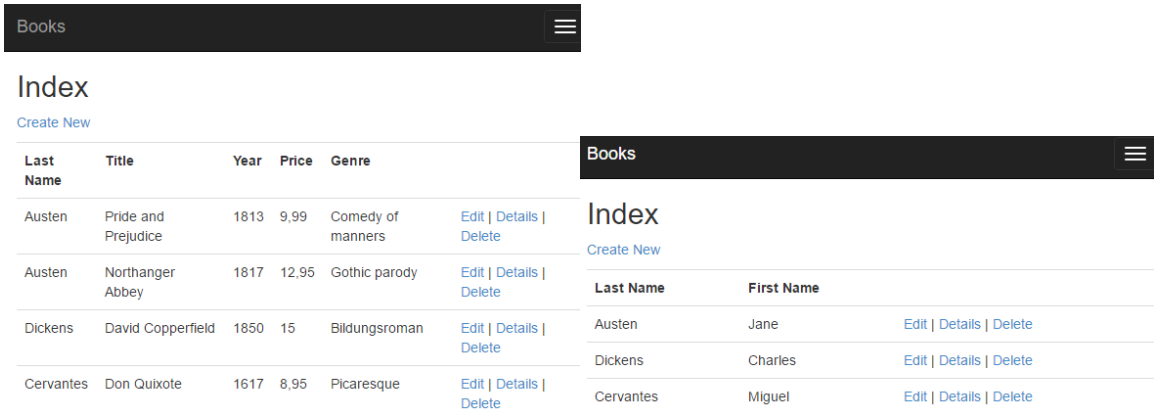

Рисунок 11 Приклад роботи програми

Завдання

1. Ознайомитися з інструкціями до виконання роботи.
2. Обрати завдання відповідно до варіанту (табл. 3).
3. Згідно до інструкцій створити веб-додаток, використовуючи технологію ASP.NET.

База даних, з якою працюватиме додаток має складатись мінімум з двох таблиць, пов’язаних між собою. Наповнення бази даних має відповідати категорії товару, зазначеній у варіанті завдання (табл. 3), допускається вибір підкатегорії.

Таблиця 3 - Варіанти завдань до лабораторної роботи 1

Номер варіанту	Категорія товару
1	Парфумерія
2	Автомобілі
3	Годинники
4	Телефони / Смартфони
5	Меблі
6	Комп’ютери / Ноутбуки
7	Книги
8	Посуд
9	Спортивний інвентар

4. Перевірити роботу програми, зберегти скриншоти результатів.

Зміст звіту

1. Мета та зміст роботи.
2. Призначення та можливості застосування ASP.NET.
3. Результати роботи програми.
4. Висновки по роботі.

Контрольні запитання

1. Для чого може бути застосованим ASP.NET?
2. Які переваги застосування ASP.NET перед іншими технологіями?

3. Які обмеження з’являються при створенні додатків з використанням ASP.NET?
4. Що таке скаффолдинг?

4.3 Лабораторна робота 3

Створення додатку, що використовує технологію ADO.NET

Мета роботи: отримати навики роботи з базами даних із застосуванням можливостей технології ADO.NET.

Зміст роботи: створення простої бази даних та робота з нею з використанням засобів ADO.NET та Windows Forms.

Інструкції до виконання роботи

1. Створення нового додатку

   Відкрити Visual Studio. В меню File обрати New => Project. У вікні, що відкриється, обрати мову Visual C# та тип додатку Windows Forms Application. Ввести назву проекту та обрати його розміщення, натиснути OK.

2. Створення бази даних

   Відкрити вікно Server Explorer. Натиснути правою клавішою миші на пункт Data Connections => Add Connection.

   Дотримуючись інструкцій у вікні, що відкрилося обрати джерело даних, тип моделі даних та створити нове підключення (рис. 12).

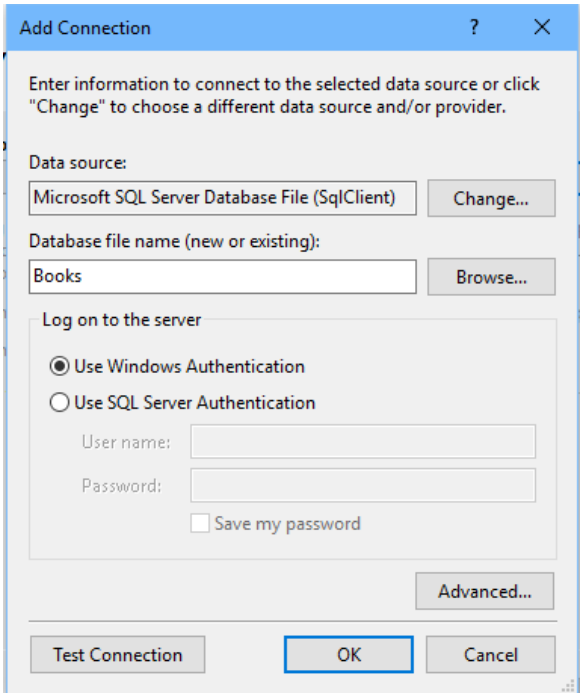

Рисунок 12 - Вибір джерела даних

Створити таблицю Books, додати такі колонки: BookId (Primary key), Title, Year, Genre, Author. Для кожної колонки обрати відповідний тип даних.

3. Використання форми для роботи з базою даних

Додати на форму елемент TableLayoutPanel, видалити останню колонку та налаштувати заповнення елементом всієї форми (*Dock: Fill*).

Додати елемент DataGridView на форму (в перший рядок TableDataLayout).

Натиснути Add Project Data Source та обрати підключення до створеної у попередньому пункті базу даних. Копіювати файл даних в проект не потрібно.

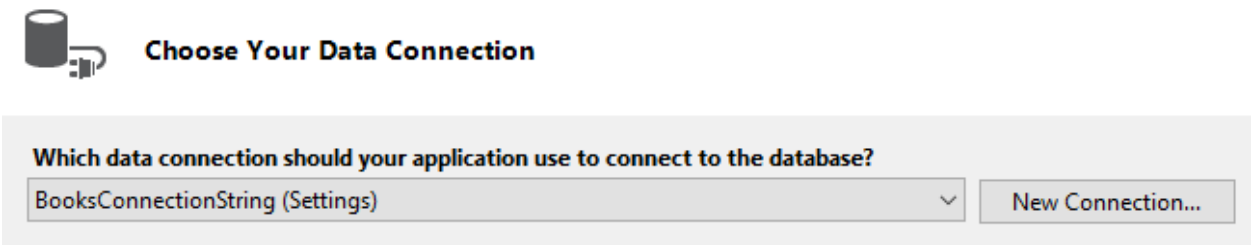

Рисунок 13 - Підключення до бази даних

Форма має прийняти наступний вигляд (рис. 14):

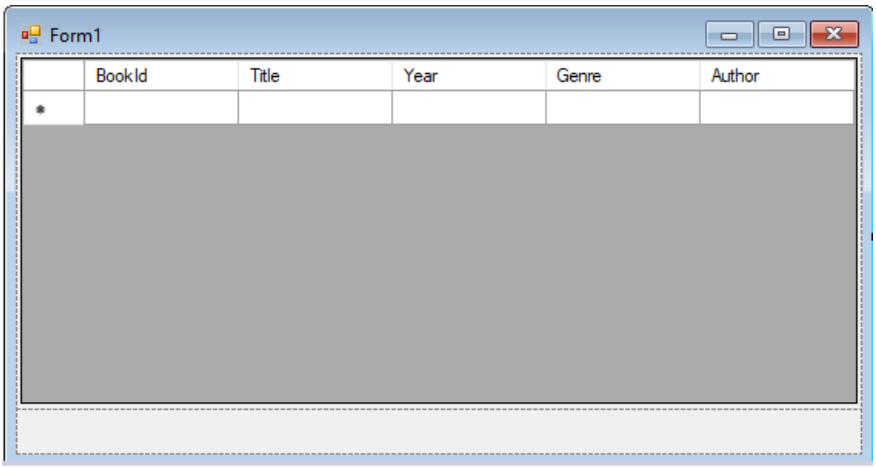

Рисунок 14 - Форма для роботи з даними

Додати у другий рядок TableLayoutPanel елемент BindingNavigator. У вікні властивостей елементу встановити властивість *BindingSource: booksBindingSource*.

Запустити додаток. Можна побачити, що є можливість введення даних у форму, але збережені вони не будуть (рис. 15).

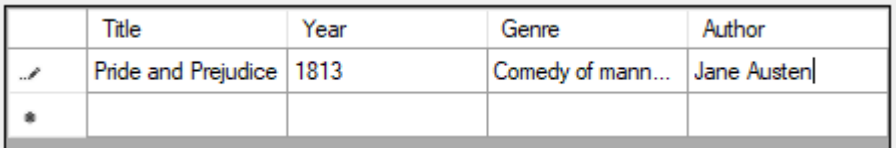

Рисунок 15 - Введення даних у форму

4. Збереження даних із форми в базі даних

Додати кнопку на *BindingNavigator*, двічі натиснути на неї для відкриття редактору коду. Аналогічно методу *Fill,* що виконується при завантаженні форми, додати в обробник натиснення кнопки метод *Update.*

Налаштувати властивість об’єкту BooksDataSet *Copy to Output: Copy if newer.*

Запустити програму; переконатися, що дані зберігаються.

Завдання

1. Ознайомитися з інструкціями до виконання роботи.
2. Обрати завдання відповідно до варіанту (табл. 3).
3. Згідно до інструкцій створити базу даних, наповнення якої буде відповідати категорії із варіанту завдання (табл. 3), та організувати взаємодію з нею через графічний інтерфейс.
4. Перевірити роботу програми, зберегти скриншоти результатів.

Зміст звіту

1. Мета та зміст роботи.
2. Призначення ADO.NET, основні об’єкти для роботи з базами даних. Особливості використання Windows Forms.
3. Результати роботи програми.
4. Висновки по роботі.

Контрольні запитання

1. Як призначення ADO.NET?
2. Які основні об’єкти включає ADO.NET для взаємодії з базами даних?
3. Для чого використовується провайдер даних?
4. Що означає поняття класів підключеного та відключеного рівня в ADO.NET?
5. Які переваги та недоліки застосування Windows Forms?

4.5 Розрахунково-графічна робота

Створити додаток, який імітуватиме роботу магазину.

Додаток має задовольняти наступним вимогам:

- загальні вимоги
  - використання складових платформи .NET, зокрема технологій ASP.NET, ADO.NET;
  - дані, що стосуються товарів, їх виробників та замовлень, мають зберігатися в базі даних (Microsoft SQL, Microsoft Access, або MySQL);
  - окрім обов’язкових атрибутів: ідентифікатору (первинного ключа), найменування (назви) та ціни, товар повинен мати як мінімум два додаткові атрибути;
  - з одним виробником можуть бути пов’язані декілька товарів;
- на стороні клієнта
  - доступ через Web-інтерфейс
  - перегляд списку наявних товарів;
  - оформлення замовлення;
- на стороні сервера
  - доступ може бути реалізований як через Web-інтерфейс, так і через WindowsForms;
  - перегляд списку наявних товарів, редагування інформації про них, видалення товару;
  - перегляд списку замовлень;

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

У даному розділі проводиться оцінка основних характеристик продукту, призначеного для ознайомлення та вивчення технологій розробки розподіленого програмного забезпечення.

Нижче наведено аналіз різних варіантів створення продукту з метою вибору оптимального, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на складність та якість викладення навчального матеріалу. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

5.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу створення навчальних матеріалів для вивчення технологій розробки розподіленого програмного забезпечення. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій продукту, призначеного для навчальних цілей.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- завдання мають бути виконуваними на персональних комп’ютерах із стандартним набором компонент;

- передбачати мінімальні витрати на встановлення програмного забезпечення, необхідного для виконання завдання.

5.2 Обґрунтування функцій продукту

Головна функція F_0– створення методичних матеріалів, які будуть допомагати у вивченні технологій розробки розподілених додатків. Виходячи з конкретної мети, можна виділити наступні основні функції продукту:

F_1 – вибір платформи для розробки;

F_2 – вибір мови програмування;

F_3 – спосіб викладення матеріалу.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) платформа.NET

б) платформа Enterprise JavaBeans;

Функція F_2:

а) мова програмування C#;

б) мова програмування Java;

Функція F_3:

а) викладений теоретичний матеріал, стисло сформульовані завдання;

б) теоретичний матеріал поєднаний із завданням, наявні приклади.

5.3 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 6). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 2).

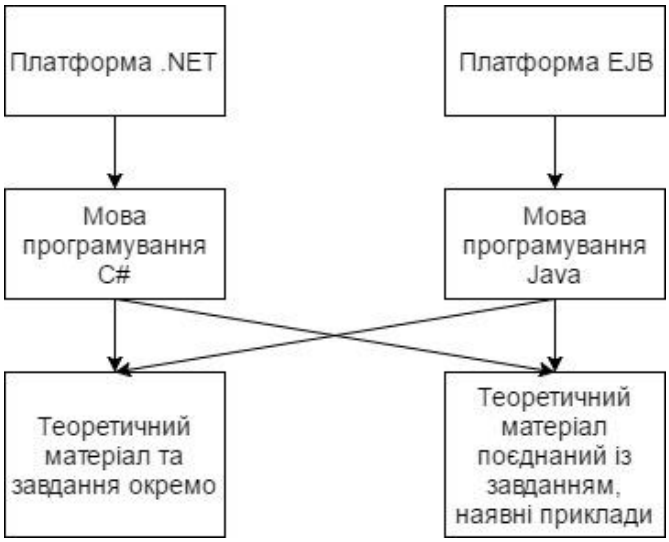

Рисунок 16 Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Можливість вибору мови програмування	Відсутність кросплатформеності
	Б	Кросплатформеність	Необхідна велика кількість ресурсів
F2	А	Більша гнучкість, інтеграція з іншими технологіями, більша швидкість	Більш молода мова, багато в чому спирається на Java
	Б	Більша кількість відкритих бібліотек	Більша складність
F3	*А*	Можливо викласти більше теоретичного матеріалу	Відсутність прикладів
	Б	Теоретичний матеріал поєднаний із завданням	Необхідно більше часу

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам.

Функція *F1:*

Завданням є створення методичних матеріалів по розробці розподілених додатків на платформі .NET, тому варіант б) відкидаємо.

Функція *F2:*

Вибір варіанта а) для F1 виключає можливість вибору варіанта б) для F2, тому вибираємо а).

Функція *F3:*

Викладення матеріалу відповідно до завдання, яке супроводжується прикладами (варіант б)), є більш зрозумілим. Такий матеріал простіше засвоюється, але варіант а) також варто розглянути.

Таким чином, будемо розглядати такі варіанти реалізації продукту:

F1a – F2a – F3a

F1а – F2а – F3б

5.4 Обґрунтування системи параметрів

Опис параметрів

На підставі даних про основні функції, що повинен мати розроблюваний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати лабораторний практикум, будемо використовувати наступні параметри:

– *X1* – складність робіт;

– *X2* – час виконання робіт;

– *X3* – потенційний об’єм програмного коду.

X1: Загальна складність лабораторних робіт.

X2: Середній час, за який можливо виконати одну роботу.

X3: Потенційний об’єм програмного коду, який має бути написаний для виконання однієї роботи.

Кількісна оцінка параметрів

Найменші, середні та найбільші значення параметрів вибираються на основі вимог до лабораторного практикуму та оцінки можливості виконання робіт, вони наведені у табл. 3.

Таблиця 5 – Основні параметри

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			найменші	**середні**	**найбільші**
Складність робіт	X1	частки одиниці	0.2	0.6	1
Час виконання робіт	X2	години	0.5	2	4
Потенційний об’єм програмного коду	X3	рядки	20	50	200

За даними таблиці 3 будуються графічні характеристики параметрів – рис. 17 – рис. 19.

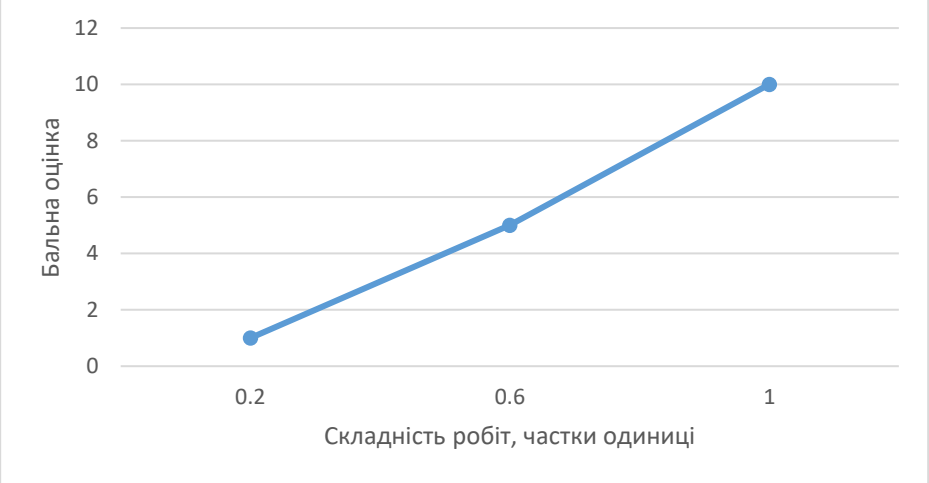

Рисунок 17 - X1 – складність робіт

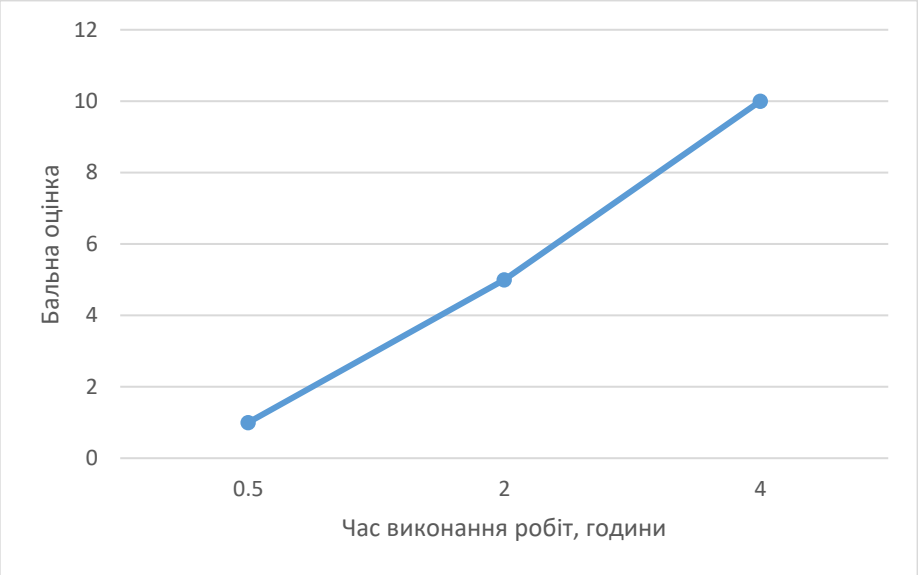

Рисунок 18 - X2 – час виконання робіт

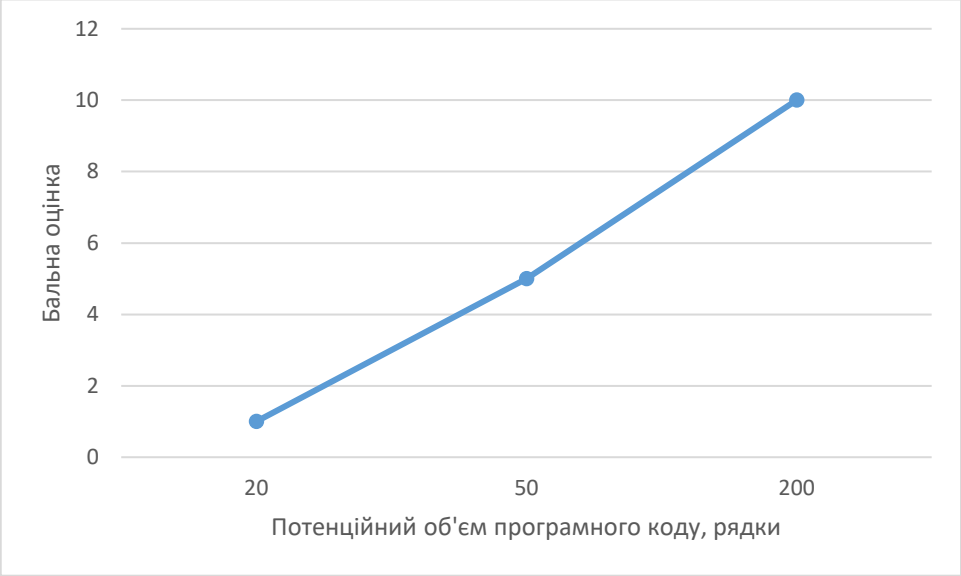

Рисунок 19 - ХЗ – потенційний об’єм програмного коду

Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – вибір реалізації лабораторного практикуму, який буде найбільш повно охоплювати можливості розробки розподілених систем, при цьому будучи зрозумілим і не надто складним.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.

Таблиця 6 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Складність робіт	частки одиниці	3	2	3	3	3	3	3	20	6	36
X2	Час виконання робіт	години	2	3	2	1	2	2	2	14	0	0
X3	Потенційний об’єм програмного коду	рядки	1	1	1	2	1	1	1	8	-6	36
	Разом		6	6	6	6	6	6	6	42	0	72

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 42,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_i = 14$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 72.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 72}{7^2(3^3 - 3)} = 0{,}73 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.

Таблиця 7 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	<	>	>	>	>	>	>	1,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	<	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5, знадобилося 6 ітерацій для того, щоб різниця значень коефіцієнтів вагомості не перевищувала 2%, більшої кількості ітерацій не потрібно.

Таблиця 8 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_i			1		2		3		4	5	6
	X1	X2	X3	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$	$K_{вi}^2$	$K_{вi}^2$	$K_{вi}^2$
X1	1	1,5	1,5	4	0,444	16,000	0,552	256,000	0,725	0,906	0,990	1,000
X2	0,5	1	1,5	3	0,333	9,000	0,310	81,000	0,229	0,091	0,010	0,000
X3	0,5	0,5	1	2	0,222	4,000	0,138	16,000	0,045	0,004	0,000	0,000
Всього:				9	1	29	1	353	1,000	1,000	1,000	1,000

5.5 Аналіз рівня якості реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1(час відповіді на запит клієнта) та X4 (Кількість функцій, доступних для користувачів) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3(об’єм оперативної пам’яті, що використовується) буде найкращим у випадку обрання у F2 варіанта Б і становитиме 14, для варіанту А це значення буде 16.

Абсолютне значення параметра X2(об’єм даних, що передається по мережі) буде найкраще при обрані варіанту А 140, а при обрані варіанту Б воно буде становити 150.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 7):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 9 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X3	100	7,2	0	0
F2	А	X3	100	7,2	0	0
F3	А	X1	0,8	7,1	0,99	7,029
		X2	0,7	6,3	0,01	0,063
	Б	X1	2	5	0,99	4,95
		X2	2,5	6	0,01	0,06

За даними з таблиці 7 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \cdots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 7,029 + 4,95 = 11,979

K_{K2} = 0,063 + 0,06 = 0,123

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки

Для визначення вартості розробки продукту спочатку проведемо розрахунок трудомісткості.

Обраний варіант включає в себе написання завдань робіт, викладення теоретичного матеріалу та написання прикладів.

Завдання за ступенем новизни відноситься до групи А. За складністю алгоритмів, які використовуються в завданні, воно належить до групи 2.

Для реалізації завдання використовується нормативно-довідкова інформація, вихідні дані та документи різного розміру і структури.

Проведемо розрахунок норм часу на виконання завдання.

Загальна трудомісткість обчислюється як

$$T_О = T_Р \cdot К_П \cdot К_СК \cdot К_М \cdot К_СТ \cdot К_СТ.М,$$

де $T_Р$ – трудомісткість розробки ПП; $К_П$ – поправочний коефіцієнт; $К_СК$ – коефіцієнт на складність вхідної інформації; $К_М$ – коефіцієнт рівня мови програмування; $К_СТ$ – коефіцієнт використання стандартних модулів і прикладних програм; $К_СТ.М$ – коефіцієнт стандартного математичного забезпечення

Для обраного варіанту реалізації завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 2, трудомісткість дорівнює: $T_Р$ =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для завдання: $К_П$ = 1.51. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для завдання: $К_СК$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_СТ$ = 1.07. Тоді, за формулою 4.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 90 \cdot 1.51 \cdot 1.07 = 145.413$ людино-днів.

$T_I = 145.413 \cdot 8 = 1163.304$ людино-годин;

Обраний варіант реалізації є достатньо трудомістким, але порівняно з іншими варіантами, він найкраще задовольняє вимогам, що висуваються.

В розробці беруть участь дві людини, одна з яких виконує основну роботу з реалізації поставленої задачі, інша визначає рамки та напрямок виконання завдання, здійснює контроль та перевірку правильності виконання. Заробітна плата працівників 5000 і 9000 відповідно. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{5000 + 9000}{2 \cdot 21 \cdot 8} = 41{,}67 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д,$$

де $С_Ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників становить:

$С_{ЗП}$ = 41,67· 1163.304 · 1.2 = 58169,85 грн.

Відрахування на єдиний соціальний внесок незалежно від ризику становить 22%::

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 58169{,}85 \cdot 0.22 = 12797{,}37$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного працівника з окладом 5000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 5000 \cdot 0{,}2 = 12000$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 12000 \cdot (1 + 0.2) = 14400$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 14400 \cdot 0{,}22 = 3168$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 29000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 29000 = 8337.5$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 29000 \cdot 0.05 = 1667.5$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4 \cdot 0.75 \cdot 0.4 \cdot 1.94 = 993.12$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 29000 \cdot 0.67 = 19430$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 14400 + 3168 + 8337.5 + 1667.5 + 993.12 + 19430 = 47996.12$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 47996.12/1706.4 = 28.13$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з виконанням поставленої задачі ведуться на ЕОМ, витрати на оплату машинного часу, для обраного варіанта реалізації, складуть:

$$С_М = С_{М\text{-}Г} \cdot Т$$

$С_М = 28.13 \cdot 1163.304 = 32723.74$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0.67$$

$С_Н = 58169.85 \cdot 0.67 = 38973.8$ грн.;

Отже, вартість розробки за обраним варіантом становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

$С_{ПП} = 58169.85 + 12797.37 + 32723.74 + 38973.8 = 142664.76$ грн.;

5.7 Вибір найкращого варіанта техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 11.979 / 142664.76 = 8{,}4 \cdot 10^{-5}$;

$К_{ТЕР2} = 0.123 / 142664.76 = 8{,}62 \cdot 10^{-7}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 8{,}4 \cdot 10^{-5}$.

5.8 Висновки до розділу

В даному розділі проведено повний функціонально-вартісний аналіз реалізації лабораторного практикуму з розробки розподілених додатків, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження виконання поставленої задачі з технічної точки зору: було визначено основні функції розроблюваного практикуму та сформовано множину варіантів їх реалізації;

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з розглянутих альтернатив, більш оптимальним є перший варіант реалізації поставленої задачі. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0{,}35 \cdot 10^{-4}$.

Цей варіант реалізації має такі параметри:

– платформа.NET;

– мова програмування C#;

– теоретичний матеріал поєднаний із завданням, наявні приклади.

Даний варіант реалізації поставленої задачі найбільш точно відповідає поставленим вимогам та буде більш простим та зрозумілим.

ВИСНОВКИ

Під час виконання дипломної роботи було розглянуто поняття розподіленої системи, основні види її архітектури, принципи, переваги та область застосування для цих видів архітектури.

Класичною архітектурою для розробки розподілених систем вважається триланкова архітектура клієнт-сервер. ЇЇ також можна назвати базовою, оскільки всі розглянуті види архітектури, окрім децентралізованої, є логічним продовженням триланкової. Серед головних переваг триланкової архітектури можна назвати гарну масштабованість, відсутність необхідності адміністрування клієнтського програмного забезпечення, можливість легко переналаштувати систему при виникненні збоїв або плановому обслуговуванні, високу безпеку та надійність систему.

В роботі проаналізовано розповсюджені технології створення розподілених додатків, а саме: технологія CORBA, технології .NET та Enterprise JavaBeans. Також було проведено порівняння технологій .NET та Enterprise JavaBeans як найбільш використовуваних та конкуруючих між собою. Обидві технології є схожими за своєю природою, підтримують можливості реалізації подібних задач та функцій, але різними способами. Технології .NET та Enterprise JavaBeans є однаково потужними, вибір між ними залежить від особливостей розроблюваної системи та області її застосування.

Більш детально було розглянуто технологію .NET, її архітектуру, призначення основних складових та можливості їх використання.

На основі розглянутих матеріалів було створено лабораторний практикум. Лабораторний практикум охоплює такі основні аспекти створення розподілених додатків: побудова архітектури додатку, робота з даними, створення інтерфейсу користувача, комунікація між компонентами додатку.

Можливість створення зв’язку між компонентами розподіленого додатку було розглянуто на прикладі використання служби Windows Communication

Foundation (WCF). Створення бази даних та робота з даними були проведені з використанням технології ADO.NET. Простий веб-додаток, що взаємодіє з базою даних, був створений із застосуванням технології ASP.NET.

ПЕРЕЛІК ПОСИЛАНЬ

1. Савельева, Е. А. Методические материалы к курсу «Технологии распределенных систем» / Е. А. Савельева. – Алматы: Научно- издательский центр КазНТУ, 2010. – 90 с.

2. Архитектурные особенности проектирования и разработки Веб-приложений [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/611/467/lecture/28784?page=1. Дата доступу: 15.05.17.

3. Поддержка разработки распределенных приложений в Microsoft .NET Framework [Електронний ресурс] – Режим доступу: http://www.intuit.ru/studies/courses/1115/177/info. Дата доступу: 17.04.17.

4. Цимбал А. А. Технологии создания распределенных систем. Для профессионалов / А. А. Цимбал, М. Л. Аншина. - Санкт-Петербург: Питер, 2003. – 576 с.

5. Построение распределенных систем на Java [Електронный ресурс] – Режим доступу: http://www.intuit.ru/studies/courses/633/489/lecture/24847. Дата доступу: 20.04.17.

6. Смертельная схватка: .NET против J2EE [Електронний ресурс] – Режим доступу: http://www.javaportal.ru/java/articles/netvsj2ee.html. Дата доступу: 25.04.17.

7. Java vs .NET: почему .NET [Електронний ресурс] – Режим доступу: http://www.shapovalov.org/news/2010-05-12-678. Дата доступу: 29.04.17.

8. J2EE vs. Microsoft.NET [Електронний ресурс] – Режим доступу: http://www-01.ibm.com/software/smb/na/J2EE_vs_NET_History_and_Comparison.pdf. Дата доступу: 3.05.17.

9. Кулямин В. В. Технологии программирования. Компонентный подход / В. В. Кулямин. - Москва: ИСП РАН, 2006. – 315 с.

10. .NET Framework [Електронний ресурс] – Режим доступу:

https://ru.wikipedia.org/wiki/.NET_Framework#.D0.90.D1.80.D1.85.D0.B8.D1.82.D0.B5.D0.BA.D1.82.D1.83.D1.80.D0.B0_.NET. Дата доступу: 19.05.17

11. Торстейсон П. Архитектура .NET и программирование на Visual C++ / П. Торстейсон, Р. Оберг. – Москва: Издательский дом «Вильямс», 2002. - 656 с.

12. Албахари Дж. C# 5.0. Справочник. Полное описание языка / Дж.Албахари, Б. Албахари. – Москва: Издательский дом «Вильямс», 2002. – 1008 с.

13. Разработка распределенных приложений на платформе Microsoft .NET Framework: Учебный курс Microsoft / С. Морган, Б. Райан, Ш. Хорн, М.Бломсма. - Санкт-Петербург: Питер, 2008. – 608 с.

14. Основы ADO.NET [Електронный ресурс] – Режим доступу: http://www.intuit.ru/studies/courses/109/109/lecture/28506. Дата доступу: 21.05.17.

15. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5. 6-е изд. / Э. Троелсен. – Москва: Издательский дом «Вильямс», 2013. - 1312 с.

16. Веб-служба [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Веб-служба. Дата доступу: 29.05.17.