Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)

(назва факультету)

(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: _____

Виконав: студент (ка) _____ курсу, групи _____
спеціальності (напряму підготовки) _____

(шифр і назва спеціальності (напряму підготовки))

_____ _____
(підпис) (прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

м. Тернопіль – 201 _

АНОТАЦІЯ

Магістерська робота на тему «Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core» Юрченка Кирила Романовича. – Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61 // Тернопіль, 2019. С. – 99, рис. – 33, табл. – 6, слайдів. – 15, додат. – 3, бібліогр. – 39.

Метою магістерської роботи є створення нового прототипу розподіленого серверу веб-додатку (сайту) , який би значно спростив реалізацію товару даного веб-магазину, а також розширив ринок, на який націлена фірма, за рахунок виходу на інтернет площадку.

Методи розробки базуються на технології .NET Core, сервер бази даних Microsoft SQL, системі контролю версій Git.

На початку звіту магістерської роботи надається загальна інформація про теоретичну і документаційну складову роботи, а саме дослідження інформації, котра стосується розробки серверного програмного забезпечення за темою магістерської роботи.

У подальших частинах звіту описується процес вибору середовища, фреймворків .NET Core та ASP.NET Core, тестування, а також процес розгортання розробленої системи на локальному сервері.

У результаті було здійснено дослідницько-інженерну роботу та завершено розробку програмного забезпечення по темі магістерської роботи.

КЛЮЧОВІ СЛОВА: СЕРВЕР, КЛАСТЕР, .NET CORE, РОЗПОДІЛЕНІ СЕРВЕРИ, БЕК-ЕНД, ВЕБ-ФЕРМА.

ANNOTATION

Master thesis on "Development of distributed server software for processing large amount of information of custom online projects based on .NET Core" by Kirill Yurchenko. - Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPm-61 group // Ternopil, 2019.

Explanatory note to the report on master's work: 99 pp., 33 fig., 6 tables., 3 appendixes, 39 sources.

The purpose of the master's thesis is to create a new prototype of a distributed server web application (site), which would greatly simplify the sale of goods of this web store, as well as expand the market targeted by the company by accessing the Internet platform.

The purpose of the master's thesis is to perform work in the subject area: "Development of distributed server software for processing large amount of information of custom online projects based on .NET Core".

Development methods are based on .NET Core technology, Microsoft SQL database server, Git version control system.

At the beginning of the master's report, general information about the theoretical and documentary component of the work is provided, namely the study of information related to the development of server software on the topic of the master's thesis.

The following parts of the report describe the process of selecting the environment, .NET Core and ASP.NET Core frameworks, testing, as well as the process of deploying the developed system on a local server.

As a result, research work was completed and software development on the topic of the master's thesis was completed.

KEYWORDS: SERVER, CLUSTER, .NET CORE, DISTRIBUTED SERVER, BACK-END, WEB-FARM.

ЗМІСТ

ВСТУП 7
1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 9
1.1 Аналіз предметної області 9
1.2 Постановка задачі 11
1.3 Виявлення основних вимог до сайту 12
1.4 Пошук акторів та варіантів використання 12
2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 16
2.1 Моделювання архітектури системи 16
2.2 Вибір процесу розробки 18
2.3 Розподілене програмне забезпечення для кластеру серверів 24
2.4 Фреймворк ASP.NET Core створений на основі .NET Core 29
2.5 Середовище розробки Visual Studio 2019 32
2.6 Побудова схеми бази даних 33
2.6.1 Виявлення основних сутностей предметної області 33
2.6.2 Нормалізація бази даних 34
2.6.3 Побудова схеми реляційної бази даних 37
2.7 Побудова UML-діаграми класів 38
3 КОНСТРУЮВАННЯ ПРОГРАМНОЇ СИСТЕМИ 43
3.1 Вибір СУБД та опис її фізичної моделі 43
3.2 Вибір системи контролю версій 44
3.3 Вибір мови програмування 45
3.4 Реалізація деяких класів системи 46
4 ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ 57
4.1 Розгортання програмної системи та системні вимоги 57
4.1.1 Використання серверу Kestrel 58
4.1.2 Розгортання додатку на веб-сервері IIS 60
4.2 Опис типових схем використання системи 66
5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 69
5.1 План використання Unit-тестів 69

5.2 Розробка Unit-тестів 71
6 ОРГАНІЗАЦІЙНО–ЕКОНОМІЧНА ЧАСТИНА 73
6.1 Загальний підхід до визначення економічної ефективності розробки 73
6.2 Розрахунок вартості процесу розробки та оцінка економічної ефективності проекту 75
7. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 85
7.1 Охорона праці 85
7.2 Безпека в надзвичайних ситуаціях 88
7.2.1 Особливості роботи користувачів комп'ютерів 89
7.2.2 Розлади здоров'я користувачів, що формуються під впливом роботи за комп'ютером 91
ВИСНОВКИ 94
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 96
ДОДАТКИ 100

ВСТУП

Кожен студент, котрий має намір отримати науковий ступінь магістра повинен оволодіти навичками збору інформації, її використання, навичками розробки та проектування великомасштабних проектів, а також навичками навчання інших людей, задля подальшого розповсюдження отриманих знань з іншими людьми та соціальної комунікації.

Темою магістерської роботи є: "Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core", котра була видана мені (виконавцю) для реалізації потреб у спрощені та розширені способу реалізації продукту, шляхом виходу на інтернет-покупців.

Враховуючи це, переді мною постає проблема в створенні нового прототипу розподіленого серверу сайту, який би значно спростив реалізацію товару даного веб-магазину, а також розширив ринок, на який націлена фірма, за рахунок виходу на інтернет площадку.

Якщо розглядати дане завдання детальніше, то можна виявити такі основні підпункти як: проектування, реалізації та тестування даного проекту. Оскільки персонал, котрий працює над цим проектом обмежений, тому мені необхідно зосередити весь свій час та увагу на доскональне виконання серверної половини даного проекту, що при правильно виконаній роботі є запорукою успіху всього проекту.

Під час збільшення об'єму інформації та відвідувачів сайту, сервер сайту стикається з багатьма проблемами, пов'язаними з його продуктивністю та обробною здатністю[7]. Найчастіше вони виражаються в тому, що сервер перестає справлятися з навантаженням. Існує кілька способів вирішення цього питання - оптимізація графіки і скриптів, нарощування потужності сервера, створення розподіленого сервера, що складається з декількох комп'ютерів. Кожен з методів має свої першочергові переваги, але перші два

методи мають межу вдосконалення потужності, котру можна подолати лише третім способом.

Третій спосіб підвищення продуктивності веб-сервера - використання "кластера" з двох або більше комп'ютерів. Це рішення менш популярно, ніж використання багатопроцесорних технологій. І це дивно, оскільки "кластер" має ряд переваг перед одним потужним сервером.

Основна частина магістерської роботи полягає в виконанні поставленого перед студентом завдання темою магістерської роботи. Ціллю теми була розробка розподіленого серверного програмного забезпечення на основі фреймворку .NET Core та побудованих на ньому розширеннях[8]. Було поставлено завдання розробити сайт, який би значно спростив реалізацію товару даного веб-магазину, а також розширив ринок, на який націлена фірма, за рахунок виходу на інтернет-площадку. Сайт має розміщатись на кластері серверів, що дозволить розподілити навантаження на сайт.

На сьогоднішній час більшість товарів та послуг реалізовуються через інтернет, саме тому, кожній фірмі, яка бажає розвиватися в правильному напрямку необхідно мати в наявності власний сайт, який би значно спростив реалізацію товару веб-магазину. На даний момент велика частина всіх існуючих підприємств, фірм та компаній мають власні сайти, на яких вони активно розвивають свою діяльність. Дані сайти зазвичай виконують основну роль в роботі з клієнтами, пропонують каталог товарів, реєструють замовлення, надають корисну інформацію і т.д.

Якщо розглядати поставлене переді мною завдання в цілому, то варто відмітити, що створення сайтів є мабуть одним з найбільш поширеніших видів роботи IT-спеціаліста, оскільки саме сайт дає великий спектр можливостей, від отримання всіє необхідної інформації до взаємодією з користувачем, що напряму впливає на успішність того чи іншого веб-магазину, компанії, стартапу тощо. Саме тому, варто відмітити про значну актуальність даної роботи в IT-сфері.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз предметної області

Кожному студенту було призначено тему магістерської роботи, моєю темою є "Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core".

Весь етап детального аналізу даної предметної області, можна розділити на 4 частин. Це аналіз роботи сайту, виявлення основних вимог до сайту, вибір моделі розробки, а також пошук акторів та варіантів використання.

Аналіз роботи сайту повинен описувати, як повинен працювати даний сайт. Даний аналіз повинен детально зазначити суть сайту, його призначення тощо.

Виявлення основних вимог до сайту дозволить зазначити всі основні пункти, яких потрібно дотримуватися при проектуванні даного сайту.

Вибір моделі розробки є мабуть одним із найголовніших етапів на старті будь-якого проекту, оскільки саме методологія розробки програмного забезпечення, або ж сайту, визначає життєвий цикл даного проекту. Тому, від вибору моделі розробки напряму впливає на подальшу роботу команди розробників.

Етап пошуку акторів та варіантів використання даної системи дозволить отримати чітку картину того, як повинен працювати та функціонувати даний сайт. Саме варіант використання надає список всіх функцій, що буде виконувати сайт.

Розбиття великого етапу розробки на менші частини, є дуже популярною методикою в багатьох проектах. Дана практика дуже позитивно впливає, як на термін виконання, так і якість реалізації даного проекту, оскільки кожен з учасників команди розробників, зосереджений лише на

відповідному сегменті етапу реалізації сайту. Також, даний спосіб дозволяє розділити поміж працівників весь проект на логічні частини, що дозволить простіше контролювати процес виконання роботи.

Для мене магістерська робота полягала в написанні логіки роботи сайту, за допомогою реалізації розподіленого серверного програмного забезпечення, котре буде розміщено на кількох серверах. Тобто, дане завдання ставило переді мною задачу, щоб реалізувати всі функції сайту, забезпечити їх стабільне виконання, обмін інформацією між серверами та забезпечити модульність та надійність всієї системи. Також, варто згадати, що оскільки сайт буде взаємодіяти з базою даних, тому забезпечення даного зв’язку також входить до моєї магістерської роботи.

Від того, як буде виконано дана магістерська робота, напряму залежить успішність та тривалість проекту, оскільки саме дане завдання, полягає в тому, щоб зв’язати всі інші частини етапу реалізації[7]. Тобто пов'язати разом базу даних та дизайн сайту, і щоб все це правильно та стабільно працювало відповідно до описаних вимог.

До списку задач, що входять до даної магістерської роботи можна віднести:

Створення архітектурного шаблону - MVC. Хоча доступно багато вже реалізованих та доступних для вільного використання подібних шаблонів, варто зазначити, що написання власного MVC шаблону дозволить без проблем в майбутньому вносити зміни до сайту.

Для реалізації MVC шаблону буде використано фреймворк ASP.NET Core, побудований на основі фреймворку .NET Core.

Ядро ASP.NET було повністю оновлено[8]. Поєднавши .NET Core та ASP.NET було досягнуто модульної структури проекту. Віднині, розробник додає індивідуальні пакети NuGet для всіх своїх потреб. Якщо він хочете використовувати MVC, він додасть пакет NuGet для MVC (слід зауважити, що Web API тепер є частиною MVC[6]. Створюючи новий проект, з ним набагато чіткіше і простіше працювати. Dependency injection наразі ж є

вбудованою і навколо неї все побудовано. Коли потрібно використовувати якийсь інструмент та його послуги, додаєте пакет NuGet і використовуєте один із методів розширення, щоб додати пакунок до контейнера DI фреймворку ASP.NET Core.

Платформа розробки .NET Core та менеджер модульних пакетів NuGet надає можливості по розробці модульного розподіленого серверного програмного забезпечення з можливістю реалізації MVC шаблону розробки веб-додатків[6]. Розподілення інформації розміщених в базах даних на кількох серверах дозволить розробити алгоритми більш швидкого пошуку та зберігання інформації.

1.2 Постановка задачі

Завданням магістерської роботи є проектування розподіленого серверного ПЗ веб-магазину. Одним із головних вимог до продукту є максимально простий та зрозумілий інтерфейс, що дозволить любому користувачу не залежно від навичок володіння інформаційними інструментами безперешкодно користуватися програмним забезпеченням. Оскільки, даний продукт буде використовувати базу даних для збереження усієї інформації, тому потрібно забезпечити надійне з’єднання та довготривалу безперебійну роботу програми з можливістю проведення профілактичних робіт окремих серверів, відключення котрих не призведе до падіння сайту. Система повинна гарантувати цілісність усієї збереженої інформації незалежно від дій користувача, тобто необхідно обмежити функціонал користувача від внесення змін до даних.

Проаналізувавши предметну область, можемо виділити перелік задач, які необхідно вирішити:

Проектування бази даних;

Розробка архітектури системи;

Реалізація системи.

До основних функцій системи можна віднести : здійснення пошуку по ключовим словам згідно запиту користувача, оформлення замовлення товарів, добавляння товарів і список "бажаного", перегляд списку товарів, перегляд короткої, достовірної інформації щодо товарів, тощо.

1.3 Виявлення основних вимог до сайту

Щодо вимог до даного сайту, то опираючись на попередній пункт можна зробити наступний висновок про основні вимоги, які будуть поставлені перед студентом.

Серед основних вимог до даного сайту можна відмітити: створити розподілене серверне програмне забезпечення, забезпечити модульність користувацького проекту, максимально якісно відобразити наявний на складі товар, у вигляді детально описаної інформації, що дозволить користувачеві здійснити правильний вибір; реалізувати сайт простим і саме головне зручним у використанні, щоб процес перегляду списку товарів на сайті нагадував перегляд звичайного каталогу товарів, без зайвих та непотрібних пропозицій і надлишкової інформації. Також, варто до списку основних вимог варто додати і можливість перегляду та обробки всіх замовлень, які надходять від користувачів даного сайту.

Отже, таким чином, ми виявили основні вимоги до розроблюваного сайту. Опираючись на дані вимоги, нам необхідно виконати детальний аналіз, та виявити основних акторів та варіанти використання.

1.4 Пошук акторів та варіантів використання

Оскільки темою магістерської роботи є "Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core", що охоплює широкий спектр імплементації серверного ПЗ, слід визначити

варіанти використання програмної системи сайту у реалізації сайту фірми, котра потребує веб-магазин.

Ще раз детально проаналізувавши вимоги до сайту, а також принцип роботи сайту, було виявлено двох акторів – Клієнт та Адміністратор, що представлені на рисунку 1.4.1.

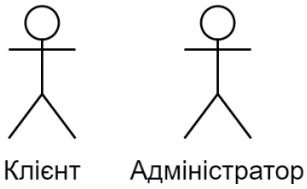


Рисунок 1.4.1 – Актори системи

Короткий опис акторів представлений в таблиці. 1.4.1.

Табл. 1.4.1 Виявлення акторів

Актор	Короткий опис
Клієнт	Перегляд списку товарів, що реалізуються даним веб-магазином, оформлення списку купівлі а також замовлення.
Адміністратор	Робота із оформленими замовленнями, категоріями товарів, інформацією, їх обробка та зміна статусу, додавання та видалення товару.

Я проаналізував роботу двох акторів в системі, та виявив наступний набір прецедентів, який наведено в таблиці 1.4.2.

Табл. 1.4.2 Виявлення варіантів використання

Основний актор	Найменування	Формулювання
Клієнт	Переглянути список товарів	Дозволяє клієнтові переглянути список наявних товарів на сайті.
Клієнт	Переглянути категорії товарів	Клієнт має змогу переглянути список категорій, на які поділені всі товари, що реалізуються на даному сайті.

Клієнт	Переглянути інформацію про окремий товар	Надає можливість клієнтові переглянути інформацію про окремий товар, його фотографію.
Клієнт	Додати обраний товар до кошика	Дозволяє додати обраний товар до кошика, щоб в подальшому оформити покупку даного товару.
Клієнт	Переглянути список товарів у кошику	Надає можливість клієнтові переглянути список товарів, що знаходяться в кошику.
Клієнт	Оформити замовлення	Дозволяє клієнтові оформити замовлення, та завершити купівлю обраних ним товарів.
Клієнт	Оплатити покупку	Клієнт може вибрати метод оплати та оплатити товар з допомогою веб-банкінгу.
Клієнт	Реєстрація	Надає можливість клієнтові зареєструвати аккаунт на сайті
Клієнт	Редагування інформації в профілі кабінету	Клієнт має можливість змінити чи додати нову інформацію в особистому кабінеті.
Клієнт	Доступ до особистого кабінету	Клієнт має можливість авторизації з допомогою особистого кабінету.
Адміністратор	Переглянути список замовлень	Адміністратор має змогу переглянути весь список замовлень, які були оформлені раніше.
Адміністратор	Переглянути окреме замовлення	Надає можливість адміністратору сайту обробляти окреме замовлення, переглядаючи інформацію про дане замовлення.
Адміністратор	Змінити статус замовлення	Дозволяє адміністратору змінити статус замовлення, безпосередньо його опрацювання.
Адміністратор	Керування категоріями	Адміністратор має змогу змінювати, додавати, видаляти категорії товарів.
Адміністратор	Керування контентом веб-сторінок	Адміністратор може створювати опитування, додавати нові рекламні слайдери на сторінки сайту.

Адміністратор	Зміна стану сайту	Адміністратор може закрити магазин для покупок.

Всі варіанти використання приведені на рисунках 1.4.2 та 1.4.3.

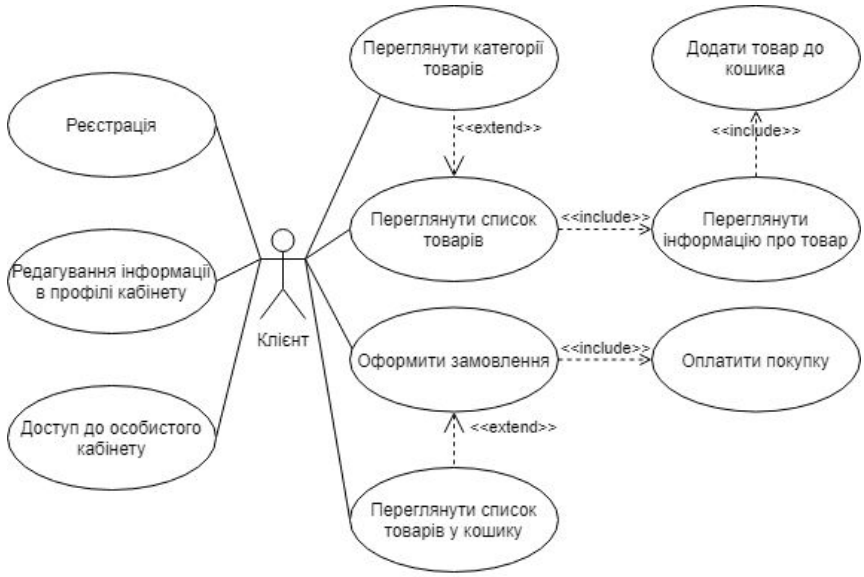


Рисунок 1.4.2 - Діаграма варіантів використання системи для актора Клієнт

Рисунок 1.4.3 - Діаграма варіантів використання системи для актора Адміністратор

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Моделювання архітектури системи

Реалізація розподіленого ПЗ потребує глибокого планування масштабованості серверної системи, і перше що приходить на думку під час масштабування ПЗ є модульність. Саме тому було обрано модульну архітектуру системи для реалізації серверного програмного забезпечення, де кожен сервіс - це модуль усієї системи.

Модуль - певна частина програми, котра містить завершену реалізацію бізнес-логіки та може використовуватись в інших програмах.

Модульне програмування - це техніка розробки програмного забезпечення, яка наголошує на поділі функціональності програми на незалежні взаємозамінні модулі, так що кожен містить усе необхідне для виконання лише одного аспекту потрібної функціональності.

Інтерфейс модуля виражає елементи, які надаються та вимагаються модулем. Елементи, визначені в інтерфейсі, виявляються іншими модулями. Реалізація містить робочий код, який відповідає елементам, оголошеним в інтерфейсі. Модульне програмування тісно пов'язане із структурованим програмуванням та об'єктно-орієнтованим програмуванням, усі вони мають однакову мету сприяти побудові великих програмних систем та систем шляхом їх розкладання на більш дрібні шматки, і всі вони починаються з 60-х років. Хоча історичне використання цього терміну було суперечливим, тепер "модульне програмування" відноситься до декомпозиції коду всієї програми на високому рівні на частини, котрі містять в собі певну споріднену алгоритмічну чи бізнес цінність.

Важливою особливістю розподіленого сервера є гнучкість. Адже в нього можуть входити машини з різним апаратним забезпеченням, операційними системами та програмним забезпеченням. На перший погляд,

здається, що це не потрібно. Насправді, ця особливість дуже допомагає при виборі операційної системи і ПО для сервера. Досить встановити на різні машини різні варіанти і простежити за їх працездатністю. Крім того, гнучкість дозволяє не прив'язуватися до комп'ютерного обладнання певної фірми і в майбутньому здійснювати дешевші апгрейди. Адже ті пристрої, які сьогодні є найсучаснішими і дорогими, через рік будуть набагато дешевше. Ну, а якщо ціна "заліза" не грає великої ролі, то власники серверу можуть часто оновлювати свої машини, маючи, таким чином, постійно покращуваний розподілений сервер.

Крім того, використання одного сервера залишає відкритим питання проведення профілактичних робіт. Адже в цьому випадку комп'ютер доведеться вимикати. Таким чином, сайт під час проведення робіт залишається недоступним. Зменшити потенційні втрати від перерв можна тільки одним способом: відключати сервер ночами, в період мінімальної відвідуваності. Однак це навряд чи сподобається технічного персоналу, який обслуговує комп'ютери. А у випадку з розподіленим сервером такої проблеми просто не існує. Машини можна зупиняти по черзі, так що відвідувачі сайту в найгіршому випадку помітять лише деяке уповільнення роботи.

Але, звичайно ж, не буває нічого ідеального. У розподіленого сервера, в порівнянні з багатопроцесорним, є один істотний мінус. Він полягає в тому, що за кожен комп'ютер, встановлений у хостинг-провайдера, доведеться платити додаткові гроші. Звичайно ж, це дуже серйозний недолік. Проте він цілком покривається плюсами розподілених серверів, а тому використання цього варіанту для серйозних проектів з великою аудиторією є більш ніж виправдано.

2.2 Вибір процесу розробки

Насамперед, слід взяти до уваги розміри цього проекту. Оскільки вимагається розробка серверної частини веб-сайту магазину, при тому що серверне ПЗ має бути розподіленим, тобто в результаті має бути розроблена веб-ферма (кластер серверів), слід вибирати з методів ітераційних методів розробки, оскільки наперед відомо, що будуть необхідні постійні виправлення і правки коду.

Раціональний об'єднаний процес (RUP) - гнучкий метод розробки програмного забезпечення, в якому життєвий цикл проекту, або розробка програмного забезпечення, поділяється на чотири фази. Під час цих етапів проводяться різні заходи: моделювання, аналіз та проектування, впровадження, тестування та застосування.

Раціональний єдиний процес (RUP) є ітераційним, тобто повторюваним; і гнучким. Ітеративний, тому що всі основні заходи процесу повторюються протягом проекту. Процес є гнучким, тому що можна регулювати різні компоненти, а фази циклу можна повторювати, поки програмне забезпечення не задовольнить вимоги та завдання.

Процес, як це зображено на рисунку 2.2.1 , слід розглядати з двох вимірів. По-перше, є часовий вимір, представлений горизонтальною віссю. Часовий вимір виражається у фазах і циклах, ітераціях та віхах. Вертикальна вісь - це розмір процесу. Цей вимір представляє статичний аспект процесу і описується в плані діяльності, артефактів, працівників та робочого процесу.

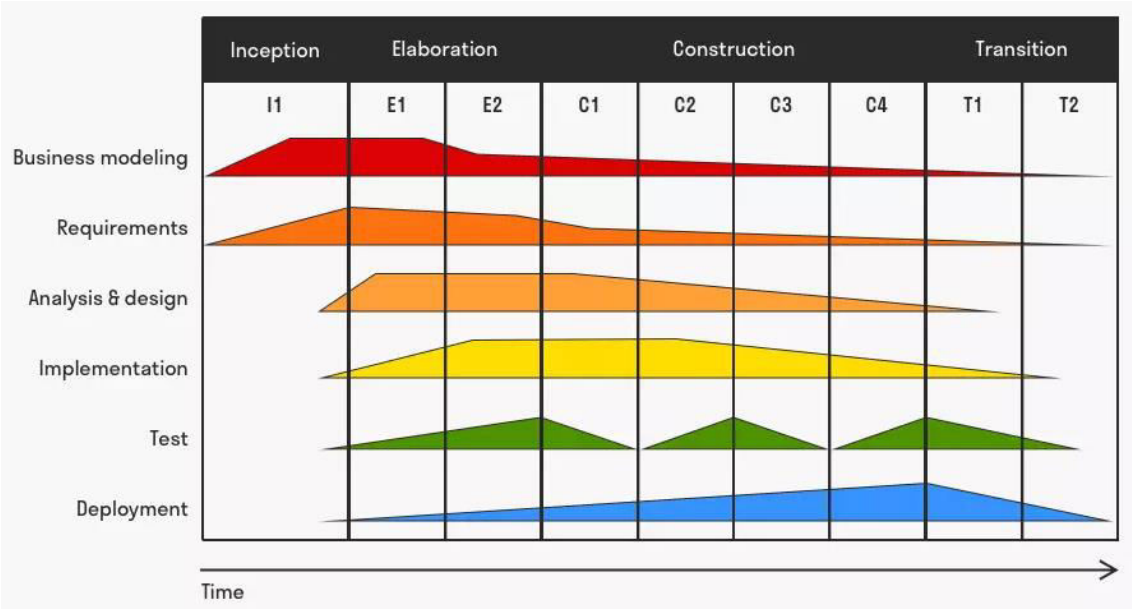


Рисунок 2.2.1 – Життєвий цикл RUP методології

Часовий вимір означає динамічну організацію процесу в часі. Життєвий цикл програмного забезпечення сам по собі розділений далі на цикли. Кожен цикл відповідає, наприклад, періоду, в якому працює над новим поколінням продукту. Раціональний єдиний процес (RUP) розділяє розвиток на чотири послідовні фази:

- Початкова фаза (Inception phase)
- Етап розробки (Elaboration phase)
- Етап будівництва (Construction phase)
- Фаза переходу (Transition phase)

Кожна фаза завершується віхою. Віхою є момент часу, коли необхідно приймати рішення, що мають критичне значення. Щоб мати можливість приймати ці рішення, цілі повинні бути досягнуті. Наприклад, віхою з перших двох етапів є прогрес реалізації варіанту використання. Варіант використання - це опис поведінки системи та дескриптор опису хто може використовувати систему і як само. Це важливий компонент у розробці програмного забезпечення. Як видно також із візуалізації RUP, тестування вже починається на першій фазі. Зазвичай продукт вже доведеться завершити до цього часу. Це тому, що тут пов'язані прототипи та тестові моделі.

Під час першого етапу визначається основна ідея та структура проекту. На цьому етапі команда регулярно збирається, щоб визначити необхідність проекту, а також його життєздатність та придатність. Життєздатність та придатність також включають очікувані витрати та кошти, необхідні для завершення проекту після надання зеленого світла.

Залежно від проекту, результатом першого етапу може бути:

1. Подання бачення проекту
2. Визначення варіантів використання (завершено на 20%)
3. Результати дослідження ринку
4. Фінансовий прогноз
5. Оцінка ризику
6. План проекту
7. Корпоративна чи ділова модель
8. Прототипи

На етапі розробки оцінюються та аналізуються вимоги системи та її необхідна архітектура. Тут проект починає формуватися. Мета етапу розробки - проаналізувати продукти та закласти основу майбутньої архітектури. Результати етапу опрацювання включають:

1. Визначення варіантів використання (на 80% виконано)
2. Опис можливої архітектури
3. План розвитку проекту
4. Прототипи для подолання ризиків
5. Посібник користувача

На етапі побудови Раціонального Уніфікованого Процесу (RUP) програмна система побудована в повному обсязі. Акцент робиться на розробці компонентів та інших особливостей системи. Більшість кодувань також відбувається в цій фазі. У цьому виробничому процесі акцент робиться на управлінні витратами та коштами, а також на забезпеченні якості. Результати на етапі виробництва включають:

1. Повністю завершена програмна система

2. Посібник користувача

Мета перехідної фази - передати товар новому користувачеві. Як тільки користувач починає користуватися системою, майже завжди виникають проблеми, які вимагають внести зміни в систему. Мета, однак, - забезпечити позитивний і плавний перехід до користувача. Результати та заходи на останній фазі:

1. Бета-тестування
2. Перетворення існуючих баз даних користувачів
3. Навчання нових користувачів
4. Розгортання проекту для маркетингу та дистрибуції

Таким чином можна ясно виділити фази, пов'язані з розробкою програмних систем. Як і в будь-якому іншому процесі, RUP описує, хто робить що, де і коли. "Хто" в цьому процесі - це працівник, який активно займається побудовою системи. "Що" означає щось конкретне, інформацію. Ці "артефакти" можуть мати різні форми, наприклад, варіант використання або прототип.

Різні фази вже вказують на різні види діяльності, що наявні у розробці системи. Далі йде більш детальне пояснення основних видів діяльності.

1. Корпоративне моделювання

Одна з проблем використання технічних систем полягає в тому, що система та користувач не мають змоги належним чином спілкуватися. Це призводить до неефективності в декількох областях. Наприклад, вхідна інформація, яку розробник отримує від користувача, неправильно використовується для розвитку генерації систем. Раціональний об'єднаний процес (RUP) частково вирішує цю проблему, створюючи універсальну мову та пропонуючи процедури рішень.

2. Вимоги

Мета вимог - описати, що система повинна робити і як вона повинна функціонувати. І користувач і розробник повинні погодитися з вимогами,

описаними на першій фазі. Все включено у документ бачення. Після цього розробляються варіанти використання.

3. Аналіз та проектування

Мета аналізу та проектування - показати, як система реалізується на етапі впровадження. Вона повинна відповідати всім вимогам, бути надійною та виконувати всі свої завдання, як описано у варіантах використання. Ця модель дизайну функціонує як креслення для решти процесу.

4. Впровадження

Впровадження відбувається протягом усього Раціонального об'єднаного процесу (RUP), як і будь-яка інша діяльність, але це є однією з інженерних дисциплін моделі. Мета реалізації - побудувати повну систему. Саме тут компоненти перевіряються та випускаються.

5. Тестування

Мета тестування - перевірити належну інтеграцію всіх компонентів та програмного забезпечення. Фаза тестування також полягає в тому, коли виявляються та усуваються дефекти. Тестування відбувається не лише на етапі тестування. Раціональний об'єднаний процес (RUP) є ітеративним, тому тестування відбувається протягом усього проекту.

Випробування проводяться за трьома вимірами:

- Надійність
- Функціональність
- Управління додатками та продуктивність системи

6. Застосування

Мета застосування системи - це, природно, успішне звільнення програмної системи та надання можливості користувачеві працювати з новою системою. Він включає багато видів діяльності, описаних на перехідному етапі 4, включаючи:

- Упаковка
- Поширення
- Установка

- Довідка та допомога
- Бета-тести
- Міграція даних
- Прийняття

Крім того, є три супутні дисципліни:

7. Управління конфігурацією та змінами;
8. Управління проектами;
9. Управліня середовищем.

Переваги методології розробки програмного забезпечення RUP:

- Ця методологія робить акцент на точній документації
- Це дозволяє активно вирішувати проектні ризики, пов'язані з вимогами, що розвиваються клієнтами для ретельних змін та управління запитами
- Дуже менше потребує інтеграції, оскільки процес інтеграції триває протягом усього процесу розвитку

Недоліки методології розробки програмного забезпечення RUP:

- Розробник програмного забезпечення повинен бути досвідченим у своїй роботі над розробкою програмного забезпечення за цією методологією.
- Процес розробки в цій методології дуже складний і не точно організований.
- Інтеграція протягом усього процесу розробки програмного забезпечення додає плутанини, яка викликає більше проблем на етапах тестування.
- Цей процес занадто складний, тому його важко зрозуміти.

2.3 Розподілене програмне забезпечення для кластеру серверів

Оскільки суттю магістерської роботи є розробка розподіленого серверного ПЗ, то слід насамперед визначитись як саме ПЗ буде розподілятись і на яких обчислювальних машинах[29]. В даному випадку, оскільки розробляється сайт з на основі фреймворку ASP.NET Core, то дана група обчислювальних машин(комп'ютерів) буде називатись кластером. У сфері веб-програмування кластер називають веб-фермою.

Більшість веб-проектів починає своє життя з невеликої кімнати і маленького сервера. Очевидно, розробники сподіваються на успіх свого дітища. Але коли приходить популярність, то разом з нею з'являються і додаткові проблеми. Найчастіше вони виражаються в тому, що сервер перестає справлятися з навантаженням. Існує кілька способів вирішення цього питання - оптимізація графіки і скриптів, нарощування потужності сервера, створення розподіленого сервера, що складається з декількох комп'ютерів. Кожен з методів має свої першочергові переваги, але перші два методи мають межу вдосконалення потужності, котру можна подолати лише третім способом.

Третій спосіб підвищення продуктивності веб-сервера - використання "кластера" з двох або більше комп'ютерів. Це рішення менш популярно, ніж використання багатопроцесорних технологій[27]. І це дивно, оскільки "кластер" має ряд переваг перед одним потужним сервером.

Кластер серверів являє собою групу з кількох машин, які об'єднані в єдиний апаратно-програмний ресурс. Для з'єднання окремих серверів в кластер застосовуються високошвидкісні канали зв'язку. Користувачі і додатки бачать їх як єдину високопродуктивну і надійну систему. Залежно від призначення виділяють три види кластерів.

Легкодоступний кластер.

Це кластер високої доступності, в якому при відмові одного сервера його функції переймають на себе інші машини в кластері. Таким чином,

сервіси та додатки продовжують працювати без зупинки завдяки апаратної надмірності. Щоб побудувати відмовостійку структуру, потрібно мінімум два фізичних сервера з системами зберігання даних. Віртуальні машини, запущені на серверах кластера, працюють за наступним принципом: якщо одна з них виходить з ладу, в роботу автоматично включається друга.

Застосування: всюди, де потрібна безперервна робота бізнес-сервісів і підтримка важливих баз даних (банк, біржа, цілодобове виробництво), електронні торгові майданчики. Переваги: надійність, висока доступність сервісів і додатків, скорочення втрат через простої в роботі.

Кластер з балансуванням навантаження.

В межах цього кластеру навантаження, яку створюють сервіси і додатки, рівномірно розподіляється між доступними машинами. Так виключається простий одного сервера, поки другий працює на межі можливостей. Для розподілу запитів в кластері використовується один або кілька вхідних обчислювальних вузлів, через які завдання перенаправляються з однієї машини на іншу.

Застосування: ЦОД, комплекси для ERP / CRM-систем, сервіси білінгу в телекомунікаційних системах. Переваги: масштабованість, можливість використання недорогих серверів стандартної архітектури[30].

Обчислювальний кластер.

Кластер побудований на основі декількох серверів, об'єднаних високошвидкісними лініями передачі інформації і спеціальним ПО. На виході утворюється єдина система для комплексних обчислень. Кожен сервер в такому кластері обробляє завдання, яка автоматично виділяється йому із загального обсягу роботи.

Застосування: аналітика, збір і обробка даних для Big Data, системи штучного інтелекту, нейронні мережі. Переваги: висока продуктивність в ресурсномістких завданнях, виконання паралельних обчислень.

Також існує типізація кластерів за "Класичною таксономією Фліна".

Таксономія Фліна розрізняє багатопроцесорні комп'ютерні архітектури відповідно до того, як їх можна класифікувати за двома незалежними вимірами Інструкційний потік та Потік даних . Кожен з цих вимірів може мати лише один з двох можливих станів: одинарний або множинний . Матриця нижче визначає 4 можливі класифікації відповідно до Фліна[30]:

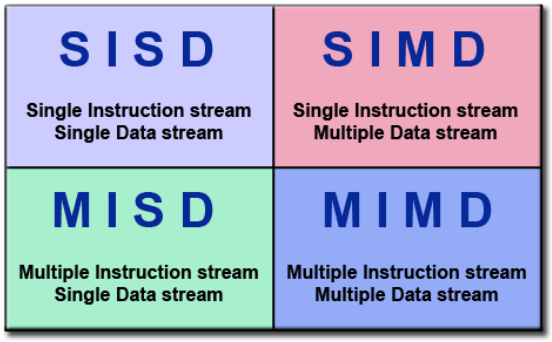
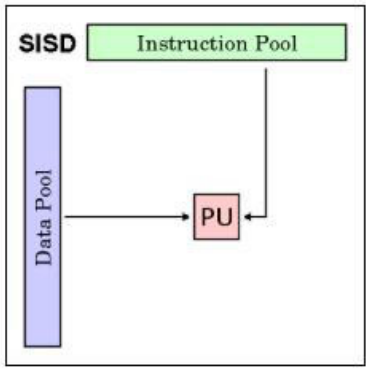


Рисунок 2.3.1 - Матриця класифікації кластерів за Фліном.

Єдина інструкція, єдині дані (SISD) (Рисунок 2.3.2):

- Послідовний (непаралельний) комп'ютер
- Єдина інструкція: центральний процесор працює протягом одного циклу часу лише одним потоком інструкцій
- Одиничні дані: лише один потік даних використовується як вхід під час будь-якого циклу годин
- Детерміноване виконання
- Це найдавніший тип комп'ютера
- Приклади: мейнфрейми старшого покоління, мінікомп'ютери, робочі станції та однопроцесорні / основні ПК.



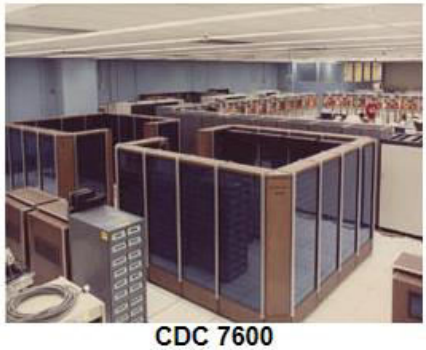

CDC 7600

Рисунок 2.3.2 - Схема роботи кластера SISD та його представник CDC 7600

Єдина інструкція, кілька даних (SIMD) (Рисунок 2.3.3):

- Тип паралельного комп'ютера
- Єдина інструкція: всі процесорні блоки виконують одну й ту ж інструкцію в будь-який заданий тактовий цикл
- Кілька даних: Кожен блок обробки даних може працювати над різним елементом даних
- Найкраще підходить для спеціалізованих проблем, що характеризуються високим ступенем регулярності, таких як обробка графіки / зображення.
- Синхронне (блокування кроків) та детерміноване виконання.
- Два різновиди: процесорні масиви та векторні трубопроводи[27].
- Приклади:
    - Маси процесора: Мислячі машини CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - Векторні: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- Більшість сучасних комп’ютерів, зокрема комп'ютери з графічними процесорними блоками (GPU), використовують інструкції та виконавчі модулі SIMD.

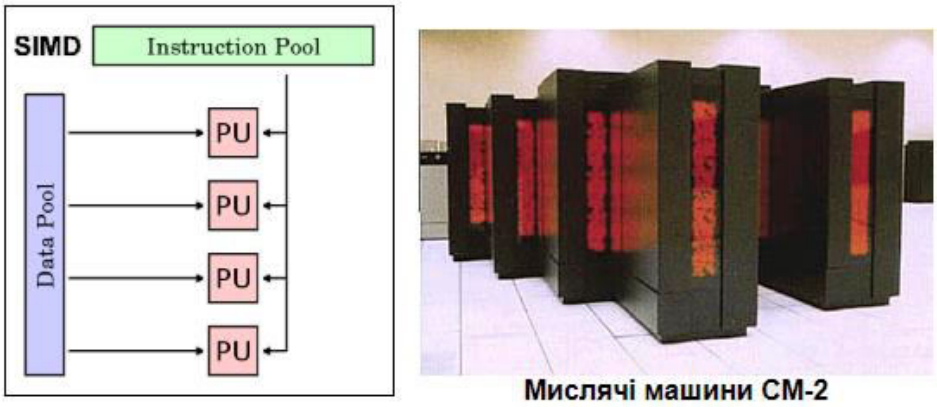


Рисунок 2.3.3 - Схема роботи кластера SIMD та його представник CM-2

Кілька інструкцій, єдині дані (MISD) (Рисунок 2.3.4):

- Тип паралельного комп'ютера
- Кілька інструкцій: Кожен блок обробки даних працює незалежно від окремих потоків інструкцій.
- Одиничні дані: Один потік даних подається в декілька одиниць обробки.
- Мало (якщо є) фактичних прикладів цього класу паралельних комп'ютерів коли-небудь існувало.
- Деякі можливі варіанти використання можуть бути:
    - багаточастотні фільтри, що працюють на одному сигнальному потоці
    - кілька алгоритмів криптографії, які намагаються зламати одне кодоване повідомлення.

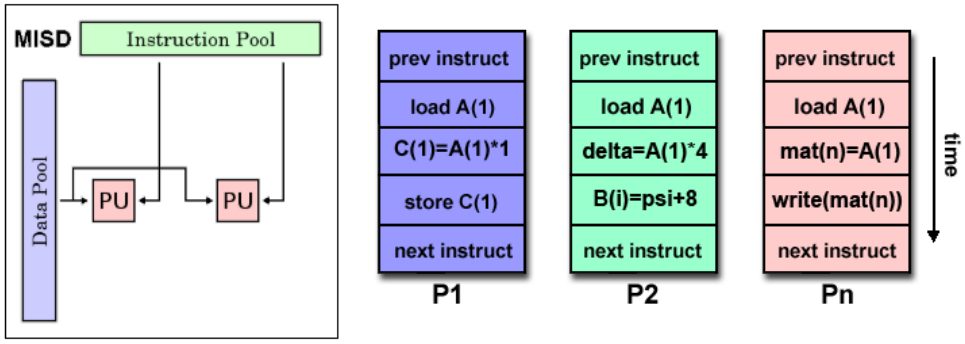


Рисунок 2.3.4 - Схема роботи кластера MISD

Кілька інструкцій, кілька даних (MIMD) (Рисунок 2.3.5):

- Тип паралельного комп'ютера
- Кілька інструкцій: кожен процесор може виконувати інший потік інструкцій
- Кілька даних: кожен процесор може працювати з різним потоком даних
- Виконання може бути синхронним або асинхронним, детермінованим або недетермінованим
- В даний час найпоширеніший тип паралельного комп'ютера - більшість сучасних суперкомп'ютерів потрапляють до цієї категорії.

- Приклади: більшість сучасних суперкомп'ютерів, мережеві паралельні комп'ютерні кластери та "сітки", багатопроцесорні комп'ютери SMP, багатоядерні ПК.
- Примітка: багато архітектури MIMD також включають підкомпоненти виконання SIMD

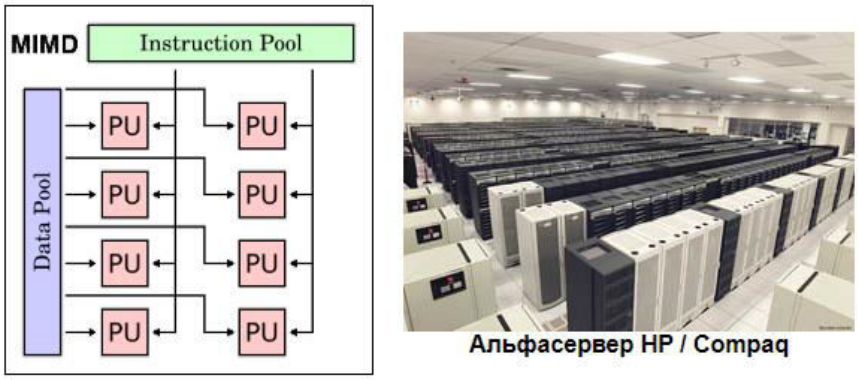


Рисунок 2.3.5 - Схема роботи кластера MIMD та кластер серверів комп. HP

2.4 Фреймворк ASP.NET Core створений на основі .NET Core

.NET Core - це модульна платформа для розробки програмного забезпечення, з відкритим вихідним кодом. Сумісна з такими операційними системами як Windows, Linux і macOS. Була випущена компанією Microsoft. У платформи є власне співтовариство на GitHub.

Підтримує наступні мови програмування: C #, Visual Basic .NET (частково) і F #.

Платформа .NET Core заснована на .NET Framework. Платформа .NET Core відрізняється від неї модульнісьтю, кросплатформеністю, можливістю застосування хмарних технологій, і тим, що в ній відбувся поділ між бібліотекою CoreFX і середовищем виконання CoreCLR.

.NET Core - модульна платформа. Кожен її компонент оновлюється через менеджер пакетів NuGet, а значить можна оновлювати її модулі окремо, в той час як .NET Framework оновлюється цілком. Кожна програма

може працювати з різними модулями і не залежить від єдиного поновлення платформи.

CoreFX - це бібліотека, інтегрована в .NET Core. Серед її компонентів: System.Collections, System.IO, System.Xml.

CoreCLR - це середовище виконання, що включає в себе RyuJIT (JIT-компілятор), вбудований збирач сміття та інші компоненти.

.NET Core та ASP.NET Core не є якимись врізаними, скаліченими версіями .NET та ASP.NET. Справа в тому, що з бази було вилучено дуже багато залежностей від Windows, але майже все, що могло б бути корисним для розробника, і що було частиною цих фреймворків, було перенесено на кросплатформений пакет NuGet.

Давайте розберемося, чому слід перейти та використовувати ASP.NET Core та ASP.NET Core MVC:

- .NET Core - це міжплатформена версія .NET, яка підтримує майже все, що підтримується .NET (крім речей, таких як WPF, Windows Forms, Web Forms та Active Directory);
- .NET Core та ASP.NET Core - безкоштовні та відкриті, але при цьому вони підтримуються Microsoft;
- І ASP.NET Core і .NET Core мають повну увагу Microsoft, і у них працюють понад 300 людей котрі працюють на ними, і це не рахуючи сотні активних учасників спільноти;
- ASP.NET Core - це веб-платформа, створена на основі платформи .NET Core. Він побудований з нуля, однак, багато старих концепцій та зразків все ще є.
- Обидва фреймворки ASP.NET Core і .NET Core легші для освоєння та роботи, ніж їх попередники.

Якщо є необхідність оновити свою систему і вона використовує веб-форми або старий MVC, розробнику слід розглянути ASP.NET Core MVC замість ASP.NET MVC 5, останнє оновлення котрого було у 2015 році.

Сотні людей з Microsoft працюють над .NET Core та ASP.NET Core! І на GitHub беруть участь багато членів спільноти!

.NET Core, ASP.NET Core, Entity Framework Core, Rosyln можуть бути відкритими та доступними на GitHub, але Microsoft надає офіційну підтримку цих продуктів і на GitHub. Однак, як і для Visual Studio, ви все одно можете взяти телефон і зателефонувати в службу підтримки будь-якого з цих продуктів. Отже, всі ці продукти підтримуються Microsoft, але .NET Core, ASP.NET Core та Entity Framework Core є абсолютно безкоштовними, з відкритим кодом та багатоплатформою, що означає: відсутність ліцензійних витрат.

Dependency injection наразі ж є вбудованою і навколо неї все побудовано. Коли потрібно використовувати якийсь інструмент та його функціонал, додаєте пакет NuGet і використовуєте один із методів розширення, щоб додати пакунок до DI контейнера фреймворку ASP.NET Core.

Конфігурація є вбудованою, а також є частиною ін'єкції залежності. Використовуйте її в будь-якому місці коду з можливістю перезавантажувати змінені значення конфігурації з джерел (appsettings.json, змінні середовища, аргументи командного рядка тощо). Також легко змінювати, розширювати та налаштовувати Конфігурацію. Немає більш масштабних конфігурацій в web.config, переважним способом зараз є appsettings.json у поєднанні із змінними оточуючих середовищ та аргументів cmd-line консолі.

Логування стало вбудованим, і ви маєте доступ до структурованих журналів від самого хоста ASP.NET Core до вашої програми. Це також легко розширити, і розробник може використовуєте різні джерела для виведення логів роботи. За допомогою таких інструментів як Serilog, ви можете легко розширити свій журнал і зберегти свої журнали до файлів Azure, Amazon або будь-якого іншого хостингу даних.

Ви налаштовуєте запит і відповідь одним методом. Це досягається за допомогою послідовної серії делегатів, які можуть або коротке замикання,

або передавати HTTP-запит наступному делегату. Вони відомі як проміжне програмне забезпечення - концепція, добре відома людям, які працювали з Node.js.

Ще одна чудова річ про ASP.NET Core - це те, що він повністю асинхронний, весь конвеєр запитів - асинхронний.

2.5 Середовище розробки Visual Studio 2019

Microsoft Visual Studio — серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework, Microsoft Silverlight, .NET Core, ASP.NET Core, Microsoft Azure (Рисунок 2.5.1).

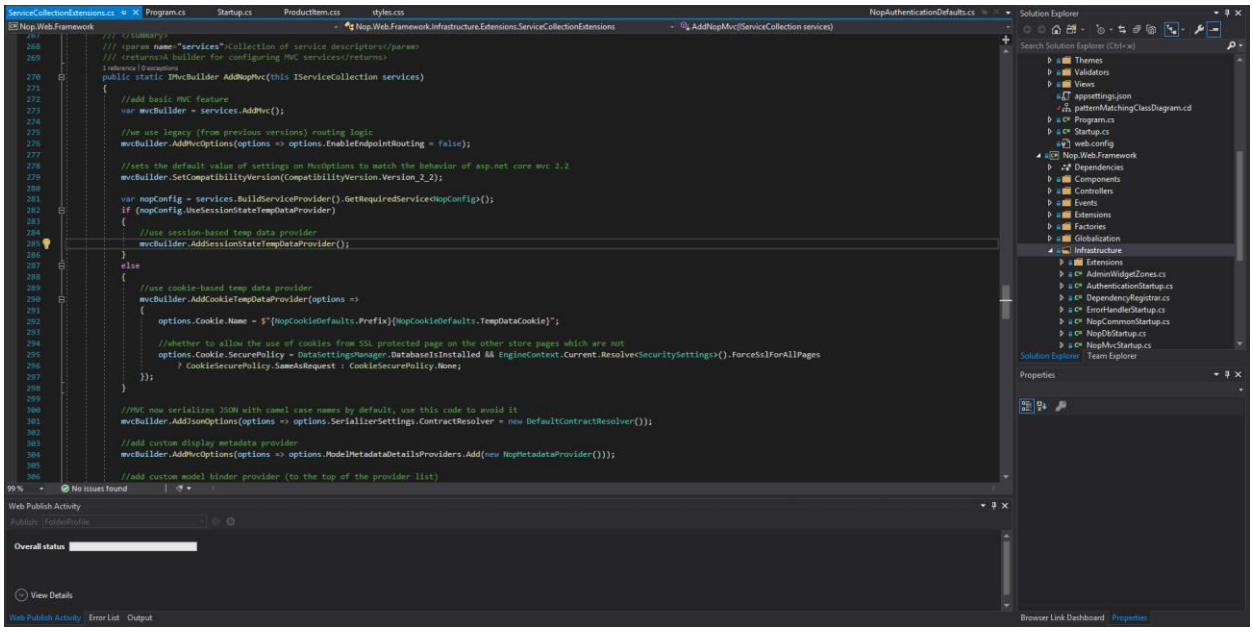


Рисунок 2.5.1 - Вікно середовища розробки Microsoft VS 2019

2.6 Побудова схеми бази даних

2.6.1 Виявлення основних сутностей предметної області

Провівши аналіз предметної області, ми можемо виявити такі сутності предметної області: Customer, Country, Activity, Product, Order, Shipment, Currency. Оскільки програмна система має велику кількість сутностей, і основною функцією програми є робота із клієнтами, то зразу можна взяти лише одну сутність для представлення у звіті. Оскільки сутність Customer має різні представлення залежно від перегляду інформації на сайті, одну велику сутність краще буде розбити на три менших сутностей бази даних – це клієнт (Customer) (Рисунок 2.6.1.1), роль клієнта (CustomerRole) (Рисунок 2.6.2.2), та адреси клієнта (CustomerAddresses) (Рисунок 2.6.2.3).

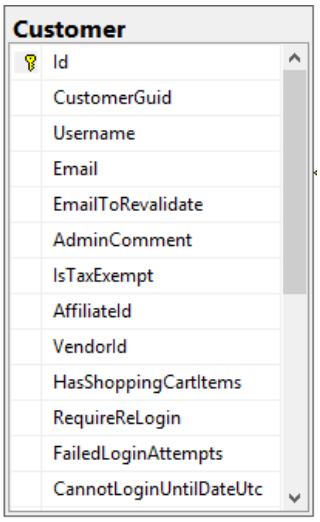


Рисунок 2.6.1.1 – Таблиця сутності Customer

Дана таблиця складається з основних атрибутів, що характеризує дану сутність. Усі поля текстового типу, оскільки всі вони будуть зберігати інформацію про клієнта.

Для таблиці CustomerRole (Рисунок 2.6.1.2) характерний такий набір атрибутів – назва, стан, звільнення від податків, чи має безкоштовну доставку. Даний набір атрибутів в повній мірі характеризує дану сутність.

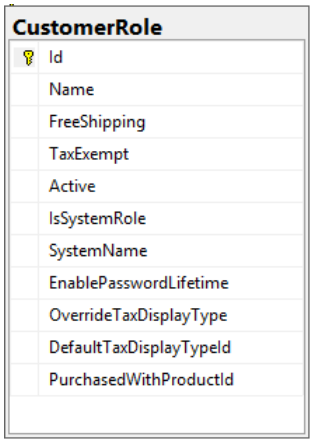


Рисунок 2.6.1.2 – Таблиця сутності CustomerRole

Таблиця CustomerAddresses (Рисунок 2.6.1.3) містить власний набір атрибутів, що характеризують дану сутність. До них належать: айді клієнта та адреси доставки. Усі поля даної таблиці також мають цілочисельний тип, оскільки всі атрибути даної таблиці складаються з чисельної інформації інформації[9]. Ця таблиця є вказівною на таблиці Address і Customer, а отже є проміжною таблицею зв'язку "багато-до-багатьох" між таблицями Customer і Address.

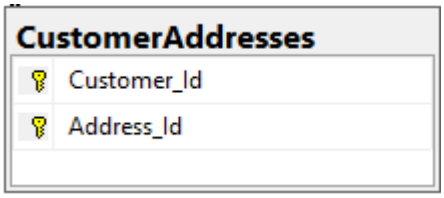


Рисунок 2.6.1.3 – Таблиця сутності CustomerAddresses

2.6.2 Нормалізація бази даних

Нормалізація - це процес організації даних у базі даних. Це включає створення таблиць та встановлення зв’язків між цими таблицями відповідно до правил, призначених як для захисту даних, так і для того, щоб зробити базу даних більш гнучкою, усуваючи надмірність та непослідовність залежності[10].

Надлишки даних витрачають дисковий простір і створюють проблеми з технічним обслуговуванням. Якщо дані, які існують у більш ніж одному

місці, повинні бути змінені, дані повинні бути змінені точно так само у всіх місцях. Зміна адреси клієнта реалізувати набагато простіше, якщо ці дані зберігаються лише в таблиці "Клієнти" і ніде-інде в базі даних[11].

Що таке "непослідовна залежність"? Хоча користувач інтуїтивно шукає в таблиці "Клієнти" адресу конкретного замовника, можливо, не має сенсу шукати там зарплату працівника, який закликає цього замовника. Заробітна плата працівника пов’язана з працівником або залежить від нього, і тому його слід перенести до таблиці працівників[12]. Невідповідні залежності можуть ускладнювати доступ до даних, оскільки шлях до пошуку даних може бути відсутнім або порушеним.

Існує кілька правил нормалізації бази даних. Кожне правило називається "нормальною формою". Якщо дотримується першого правила, то, як кажуть, база даних знаходиться у "першому нормальному вигляді". Якщо дотримуються перші три правила, база даних вважається такою, що є "третьою нормальною формою". Хоча можливі й інші рівні нормалізації, третя нормальна форма вважається найвищим рівнем, необхідним для більшості застосувань.

Як і у багатьох формальних правилах та технічних характеристиках, реальні сценарії не завжди дозволяють досконало дотримуватися[13]. Взагалі для нормалізації потрібні додаткові таблиці, і деякі клієнти вважають це громіздким. Якщо ви вирішили порушити одне з перших трьох правил нормалізації, переконайтеся, що ваша програма передбачає будь-які проблеми, які можуть виникнути, такі як зайві дані та непослідовні залежності.

Розглянемо відношення Customer-CustomerRole (Рисунок 2.6.2.1). Можна побачити, що поле id, яке характеризує код клієнта, буде відноситися до кількох записів в даному відношенні, тому слід винести даний атрибут в окрему таблицю Customer_CustomerRole_Mapping, та пов’язати їх зв’язком «багато-до-багатьох» (Рисунок 2.6.2.1).

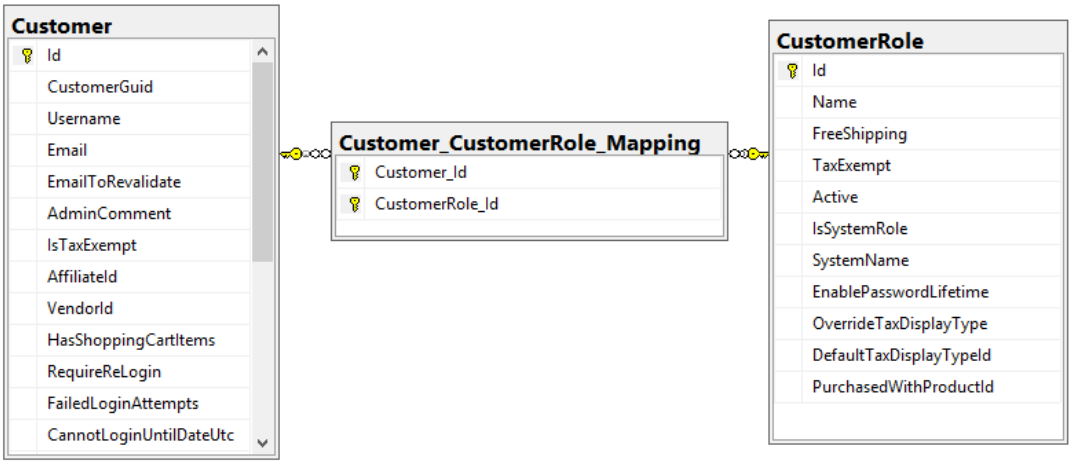


Рисунок 2.6.2.1 – Схема відношеня між таблицями Customer та CustomerRole

Для забезпечення зв’язків «багато до багатьох» між таблицями бази даних застосовуються сполучні таблиці. Вони містять первинні ключі двох таблиць, які необхідно пов’язати[14].

Помістивши в таблицю Customer_CustomerRole_Mapping зовнішній ключ, що буде вказувати на відповідний кортеж в таблиці CustomerRole, ми таким чином пов’язали ці дві таблиці зв’язком «один до багатьох».

Розглянемо відношення Customer-Address (Рисунок 2.6.2.2). На таблиці Customer можна побачити цілих два поля, значення яких можуть відноситися до кількох записів в даному відношенні до таблиці Address напряму – BillingAddress_Id та ShippingAddress_Id.

Завдяки цьому утворюються два звязки "багато-до-одного" від сутності клієнта до сутності адреси[17]. Також слід зазначити відношення "багато-до-багатьох" винесене у таблиці CustomerAddresses, котре відображає закріплену адресу за аккаунтом клієнта (Рисунок 2.6.2.2).

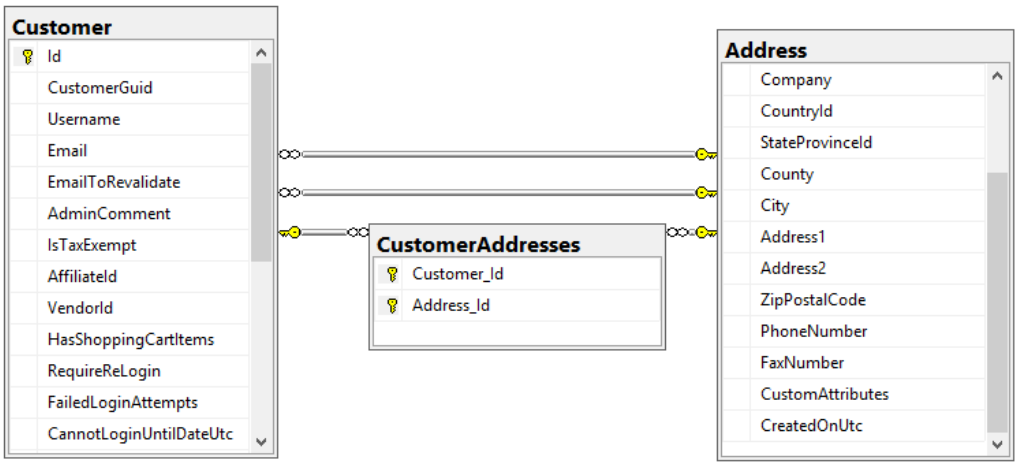


Рисунок 2.6.2.2 – Схема відношеня між таблицями Customer та Address

В результаті виконання всіх вище зазаначених замін одного відношення схемою інших, ми привели дану базу даних до 3НФ[18].

На практиці, як правило, цілком достатньо привести базу даних до 3НФ для створення надійної схеми бази даних. В результаті цього, дальше нормалізувати нашу базу даних ми не будемо.

2.6.3 Побудова схеми реляційної бази даних

Оскільки у попередніх двох розділах було вказано, що у звіті було відображено лише кілька утворених сутностей, представлених у вигляді таблиць, на загальній схемі можна відобразити усі сутності котрі стосуються клієнта[19]. Після виявлення усіх сутностей і зв'язків між ними, можна побудувати загальну схему бази даних (Рисунок 2.6.3.1).

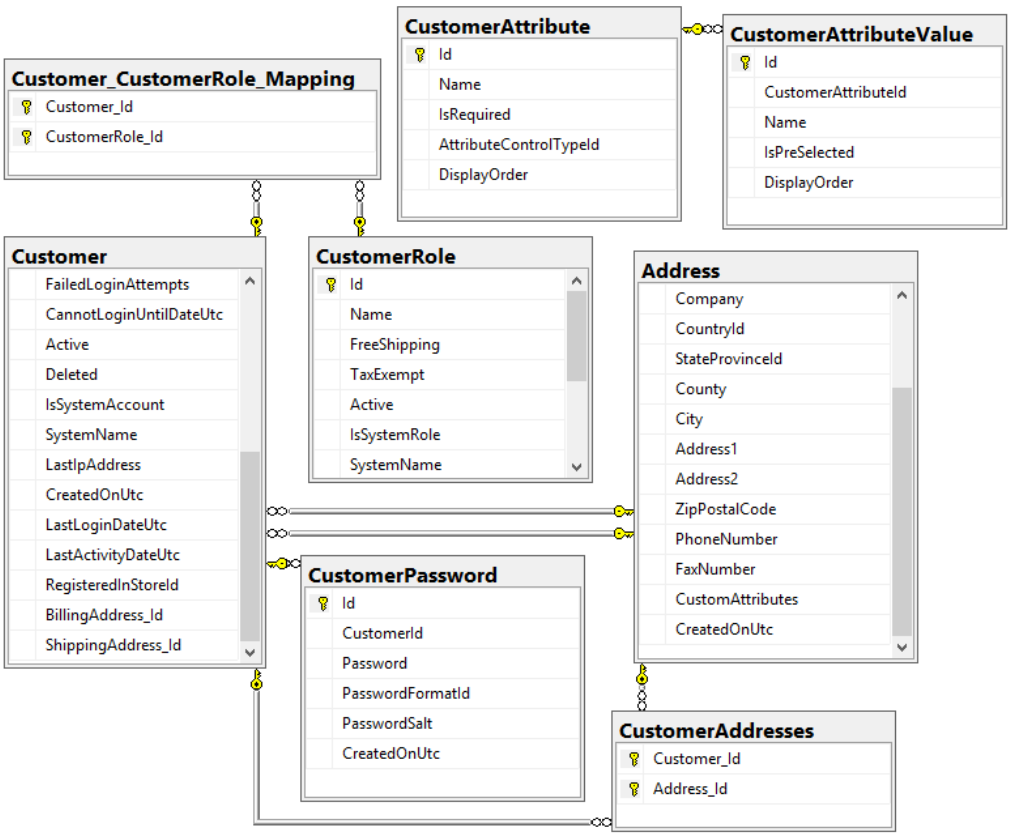


Рисунок 2.6.3.1 - Загальна діаграма сутностей котрі стосуються клієнта

2.7 Побудова UML-діаграми класів

Модель архітектури MVC розділяє додаток на три основних компоненти: Model (модель), View (подання) і Controller (контролер). За допомогою моделі MVC можна створювати додатки, які зручніше тестувати і оновлювати в порівнянні з традиційними монолітними додатками. Додатки на основі моделі MVC містять наступне:

- Моделі: класи, що представляють дані в додатку. Класи моделі використовують логіку перевірки, яка дозволяє застосовувати бізнес-правила до цих даних. Як правило, об'єкти моделі витягають і зберігають стан моделі в базі даних.
- Представлення: компоненти, які формують користувальницький інтерфейс програми. Як правило, в інтерфейсі відображаються дані моделі.

- Контролери: класи, які обробляють запити браузера. Вони витягають дані моделі і викликають шаблони уявлень, які повертають відповідь. У додатку MVC представлення служить тільки для відображення інформації. Обробку введених даних, формування відповіді і взаємодія з користувачем забезпечує контролер. Наприклад, контролер обробляє дані маршруту і значення рядка запиту, після чого передає ці значення в модель. Модель може використовувати ці значення для виконання запитів до бази даних.

Модель MVC дозволяє створювати додатки, які поділяють різні аспекти (логіка введення, бізнес-логіка і логіка призначеного для користувача інтерфейсу), забезпечуючи при цьому слабкі взаємозалежності між цими елементами. Модель визначає, в якому місці програми буде розташовуватися логіка різних видів. Логіка для користувача інтерфейсу відноситься до подання. Логіка введення відноситься до контролера. Бізнес-логіка розміщується в моделі. Подібне розділення дозволяє ефективно справлятися з труднощами при побудові програми, оскільки воно дає можливість працювати одночасно з одним аспектом реалізації, не зачіпаючи при цьому код інших. Наприклад, ви можете змінювати код представлення незалежно від коду бізнес-логіки.

До складу моделі програми ASP.NET Core MVC входять абстрактні інтерфейси і конкретні класи реалізації, що описують додатки MVC. Ця модель формується в результаті виявлення MVC контролерів, процесів, властивостей дій, маршрутів і фільтрів додатки відповідно до угод за замовчуванням. Працюючи з моделлю додатки, можна змінити додаток так, щоб для нього діяли угоди, відмінні від поведінки MVC за замовчуванням. Параметри, імена, маршрути і фільтри використовуються в якості даних конфігурації для дій і контролерів.

Структура моделі програми ASP.NET Core MVC виглядає наступним чином:

- ApplicationModel
    - Контролери (ControllerModel)
        - Дії (ActionModel)
            - Параметри (ParameterModel)

Представлення побудованих діаграм класів можна розділити на 3 типи, оскільки серверне програмне забезпечення писалося на основі фреймворку ASP.NET Core, то частина серверного ПЗ, котра генерує HTML-сторінки, опирається на великий набір класів моделей, де кожен клас несе в собі лише набір певних атрибутів без методів. Оскільки кількість класів моделей надто велика, її можна представити у згорнутому вигляді (Рисунок 2.7.1).

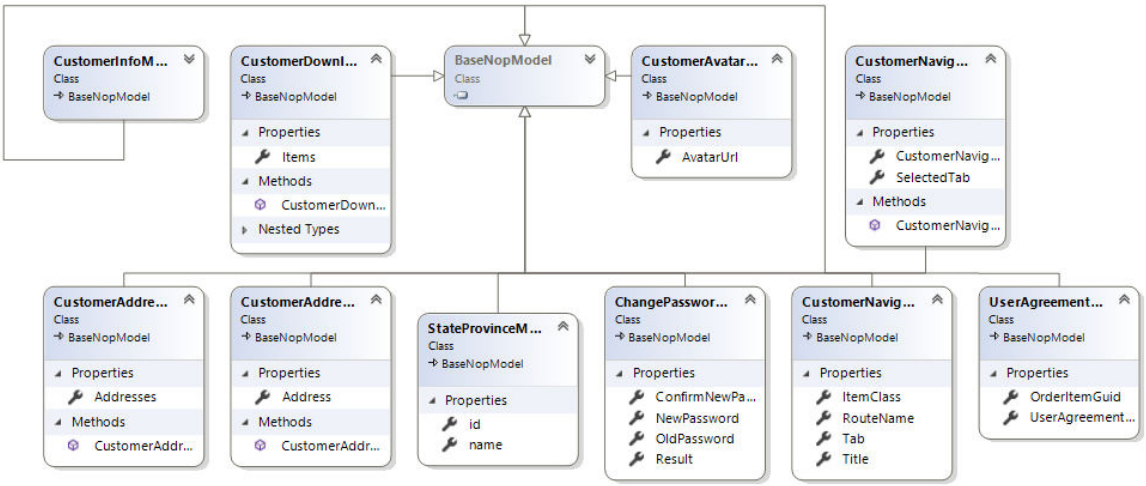


Рисунок 2.7.1 - Діграма класів моделей для зберігання інформації HTML-сторінки

Оскільки класи представлення є класами, котрі інкапсулюють в собі класи моделей і розміщують відповідні елементи моделі та компоненти у веб-сторінці, і при цьому не утворюють об'єкти, відобразити діаграму класів представлень не є можливим.

Класи контролери містять в собі методи обробки запитів, з подільшим виводом представлень у відповідності до обробки запиту користувача. Контролер використовується для визначення і угруповання набору дій. Дія (або метод дії) - це метод на контролері, що обробляє запити. Контролери

логічно групують схожі дії. Це об'єднання дій дозволяє колективно застосовувати загальні набори правил, наприклад маршрутизації, кешування і авторизації. Запити зіставляються з діями за допомогою маршрутизації.

Контролери повинні відповідати принципу явних залежностей. Існує кілька підходів до реалізації цього принципу. Якщо кільком діям контролера потрібна одна служба, рекомендується використовувати впровадження через конструктор для запиту цих залежностей. Якщо служба потрібна тільки одному методу дії, рекомендується використовувати впровадження дій для запиту залежності.

В рамках шаблону MVC контролер відповідає за початкову обробку запиту і створення екземпляра моделі. Як правило, бізнес-рішення слід виконувати всередині моделі. Для діаграми зобразимо класи контролери (Рисунок 2.7.2).

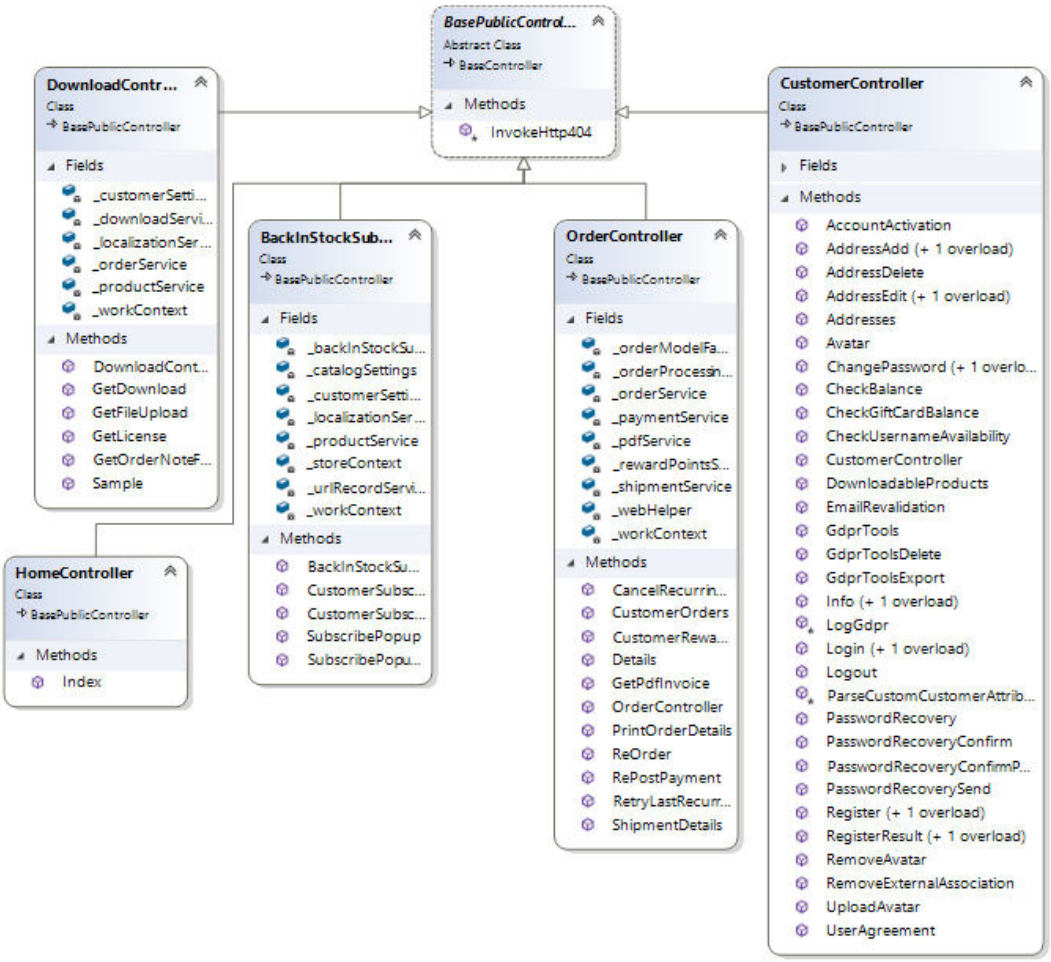


Рисунок 2.7.2 - Часткова діаграма класів контролерів у проекті

Контролер є абстракцією на рівні призначеного для користувача інтерфейсу. Він повинен перевірити допустимість даних запиту і вибрати повертається представлення (або результат для API). У добре організованих додатках він не включає в себе доступ до даних або бізнес-логіку безпосередньо. Замість цього контролер делегує обробку цих обов'язків службам.

3 КОНСТРУЮВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Вибір СУБД та опис її фізичної моделі

Для проекту була обрана СУБД Microsoft SQL. Microsoft SQL Server - це система управління реляційними базами даних (RDBMS), яка підтримує широкий спектр програм для обробки транзакцій, бізнес-аналітики та аналітики в корпоративних IT-середовищах. Microsoft SQL Server є однією з трьох лідируючих на ринку технологій баз даних, поряд з Oracle Database та IBM DB2.

Як і інше програмне забезпечення RDBMS, Microsoft SQL Server побудований на основі SQL, стандартизованої мови програмування, яку адміністратори бази даних (DBA) та інші IT-фахівці використовують для управління базами даних та запиту даних, які вони містять. SQL Server пов'язаний з Transact-SQL (T-SQL), реалізацією SQL від Microsoft, яка додає набір стандартних розширень програмного забезпечення до стандартної мови.

Як і інші технології RDBMS, SQL Server в основному побудований навколо структури на основі рядків, яка з'єднує пов'язані елементи даних у різних таблицях один з одним, уникаючи необхідності надмірно зберігати дані в декількох місцях у базі даних. Реляційна модель також забезпечує референтну цілісність та інші обмеження цілісності для підтримки точності даних. Ці перевірки є частиною більш широкого дотримання принципів атомності, узгодженості, ізоляції та довговічності, спільно відомих як властивості ACID, і розроблені таким чином, щоб гарантувати надійну обробку операцій з базами даних.

Основним компонентом Microsoft SQL Server є SQL Server Database Engine, який контролює зберігання, обробку даних та безпеку. Він включає реляційний двигун, який обробляє команди та запити, і механізм зберігання даних, який управляє файлами баз даних, таблицями, сторінками, індексами, буферами даних та транзакціями. Збережені процедури, тригери, перегляди

та інші об'єкти бази даних також створюються та виконуються двигуном бази даних.

Під двигуном бази даних знаходиться операційна система SQL Server або SQLOS. SQLOS обробляє функції нижчого рівня, такі як управління пам'яттю та введення-виведення, планування роботи та блокування даних, щоб уникнути конфліктних оновлень. Мережевий рівень інтерфейсу розташовується над Database Engine і використовує табличний протокол потоку даних Microsoft для полегшення взаємодії запитів і відповідей із серверами баз даних. І на рівні користувача, DBA і розробники SQL Server записують T-SQL заяви для створення та зміни структур баз даних, маніпулювання даними, здійснення захисту безпеки та резервного копіювання баз даних, серед інших завдань.

Також, в силу того, що Microsoft SQL є невибагливим до апаратного забезпечення, тому вибір даної СУБД є найкращим рішенням для реалізації подібного роду задач, оскільки, дана програма повинна бути встановлена на терміналах з слабкими технічними характеристиками.

Усі ці особливості були вирішальними при виборі СУБД.

3.2 Вибір системи контролю версій

Git - це набір консольних утиліт, які відстежують і фіксують зміни в файлах (найчастіше мова йде про вихідний код програм, але ви можете використовувати його для будь-яких файлів). З його допомогою ви можете відкотитися на більш стару версію вашого проекту, порівнювати, аналізувати, зливати зміни і багато іншого. Git є розподіленим, тобто не залежить від одного центрального сервера, на якому зберігаються файли. Замість цього він працює повністю локально, зберігаючи дані в папках на жорсткому диску, які називаються репозиторієм. Можливе зберігання копії сховища онлайн, це сильно полегшує роботу над одним проектом для кількох

людей. Для цього використовуються сайти на кшталт GitHub.com і GitBucket.com.

3.3 Вибір мови програмування

Для написання програми була обрана Об’єктно-Орієнтовна технологія.

Об'єктно-орієнтоване програмування (ООР) - це модель мови програмування, в якій програмна логіка організована навколо даних або об'єктів, а не функцій та логіки[1]. Об'єкт можна визначити як поле даних, яке має унікальні атрибути та поведінку. Приклади об'єкта можуть варіюватися від фізичних осіб, таких як людина, яка описується властивостями, такими як ім’я та адреса, аж до невеликих комп'ютерних програм, таких як віджети. Це протистоїть історичному підходу до програмування, де робився акцент на тому, як була написана логіка, а не на тому, як визначити дані в логіці.

Перший крок ООР - визначити всі об'єкти, якими програміст хоче маніпулювати, і те, як вони співвідносяться один з одним, вправа, часто відома як моделювання даних. Після формалізації об'єкта шляхом визначення його атрибутів і функціоналу, він узагальнюється як клас об'єктів, який визначає структуру та тип даних, який він містить, і будь-які логічні послідовності, які можуть ним керувати. Кожна окрема логічна послідовність відома як метод, і об'єкти можуть спілкуватися з допомогою чітко визначених інтерфейсів, котрі називаються повідомленнями.

Простіше кажучи, ООР зосереджується на об'єктах, якими розробники хочуть маніпулювати[2], а не на логіці, необхідній для маніпулювання ними. Такий підхід до програмування добре підходить для великих, складних і активно оновлюваних або підтримуваних програм.

Зважаючи на це, для написання даної програми розглядалися наступні ООП мови програмування: C++, Java, C# .

Серед переваг таких мов програмування як Java та C# порівняно із C++ це: «збирач сміття», який звільняє ресурси, що захищає програму від втрат пам’яті і від необхідності звільняти ресурси;

Зважаючи на ці дві вагомі переваги, C++ вибуває із переліку можливих мов програмування. Залишається Java та C#. Розглянувши ці дві мови програмування більш детально, було виявлено наступні переваги та недоліки.

Основна перевага Java на відмінну від C# є її кросплатформеність[3]. Хоч і платформа .NET заявлена як кросплатформена, для реалізації програми не під ОС Windows, необхідно використовувати додатковий проект Mono, в результаті цього, можна сказати, що .NET умовно кросплатформеним.

Продовжуючи тему кросплатформеності, варто зауважити, що дане програмне забезпечення повинно бути створене лише для однієї операційної системи. Саме тому, кросплатформеність Java, не відіграє ніякої ролі в процесі вибору мови програмування.

Також, існує особлива відмінність між даними мовами. Програми, що реалізовані мовою програмування Java, залежні від JRE. Тому, для таких програм, окрім ПЗ, потрібно також встановлювати додатково і JRE. Схожа залежність є і в C# разом із платформою .NET[7]. Проте, як правило, .NET встановлена по замовчуванню в ОС Windows.

Керуючись всіма вище зазначеними відмінностями, перевагами та недоліками, в якості програми для реалізації даного програмного забезпечення було обрано C# з технологією .NET[21].

3.4 Реалізація деяких класів системи

Оскільки програмна реалізація відбувається в рамках фреймворку ASP.NET Core, то програмісту слід реалізувати певний обов'язковий

функціонал класів системи, такі як: Startup, Program, класи моделей, контролерів, представлень, сервісів та служб.

У класі Startup робиться наступне:

- При необхідності містить метод ConfigureServices для налаштування служб додатку. Служба - багаторазово використовуваний компонент, що забезпечує функціональність програми. Служби реєструються в ConfigureServices і використовуються в додатку за допомогою впровадження залежностей (DI) або ApplicationServices.
- Містить метод Configure для створення конвеєра обробки запитів додатка.

Служби - це компоненти, які використовуються в додатку. Наприклад, службою є компонент ведення журналу. Код для налаштування (або реєстрації) служб додається в метод Startup.ConfigureServices.

Конвеєр обробки запитів складається з ряду компонентів ПО проміжного шару. Наприклад, це ПЗ може обробляти запити для статичних файлів або перенаправляти HTTP-запити в HTTPS. Кожен компонент ПО проміжного шару виконує асинхронні операції в HttpContext, а потім або викликає наступний компонент в конвеєрі, або завершує запит. Код для настройки конвеєра обробки запитів додається в метод Startup.Configure.

Реалізація класу Startup (Лістинг 3.4.1):

```
public class Startup
   {
      private readonly IConfiguration _configuration;
      private readonly IHostingEnvironment _hostingEnvironment;
      public Startup(IConfiguration configuration, IHostingEnvironment
hostingEnvironment)
      {
         _configuration = configuration;
         _hostingEnvironment = hostingEnvironment;
      }
      /// Add services to the application and configure service provider
      public IServiceProvider ConfigureServices(IServiceCollection services)
      {
```

```
            return services.ConfigureApplicationServices(_configuration,
_hostingEnvironment);
        }
        /// Configure the application HTTP request pipeline
        public void Configure(IApplicationBuilder application)
        {
            application.ConfigureRequestPipeline();
        }
}
```

Лістинг 3.4.1 - Реалізація класу Startup

Коли додаток визначає окремі класи Startup для різних середовищ (наприклад, StartupDevelopment), відповідний клас Startup вибирається під час виконання. Клас, у якого суфікс імені відповідає поточному середовищу, отримує пріоритет. Наприклад, якщо додаток виконується в середовищі розробки і включає в себе обидва класи - Startup і StartupDevelopment, використовується клас StartupDevelopment.

Використовуйте IStartupFilter в наступних випадках:

Для настройки ПО проміжного шару на початку або кінці конвеєра ПО проміжного шару Configure додатки без явного виклику Use {Middleware}. ASP.NET Core використовує IStartupFilter для додавання значень за замовчуванням в початок конвеєра, виключаючи необхідність явної реєстрації ПО проміжного шару за замовчуванням автором програми. IStartupFilter дозволяє іншим компонентам викликати Use {Middleware} від імені автора програми.

Для створення конвеєра методів Configure. IStartupFilter.Configure може забезпечувати запуск ПЗ проміжного шару до або після ПО проміжного шару, який додається бібліотеками.

IStartupFilter реалізує метод Configure, який приймає і повертає Action <IApplicationBuilder>. IApplicationBuilder визначає клас для настройки конвеєра запитів додатка. Додаткові відомості див. У розділі Створення конвеєра ПО проміжного шару за допомогою IApplicationBuilder.

Кожен інтерфейс IStartupFilter може додавати один або кілька компонентів ПО проміжного шару в конвеєр запитів. Фільтри викликаються в тому порядку, в якому вони були додані в контейнер служби. Фільтри можуть додавати ПО проміжного шару до або після передачі управління наступного фільтру, тому вони додаються в початок або кінець конвеєра додатки.

ASP.NET Core включає вбудовану платформу впровадження залежностей, що дозволяє класам додаткам звертатися до налаштованих служб. Одним з варіантів отримання екземпляру служби в класі є створення конструктора з параметром необхідного типу. Параметр може бути типом служби або інтерфейсом. Система впровадження залежностей надає службу під час виконання.

Нижче показаний клас, який використовує впровадження залежностей для отримання об'єкта контексту Entity Framework Core (Лістинг 3.4.2). Виділений рядок є прикладом впровадження через конструктор.

```
public Startup(IConfiguration configuration, IHostingEnvironment
hostingEnvironment)
{
  _configuration = configuration;
  _hostingEnvironment = hostingEnvironment;
}
```

Лістинг 3.4.2 - Впровадження залежностей через конструктор

Незважаючи на те, що система впровадження залежностей є вбудованою, вона дозволяє програмісту при бажанні підключати сторонній контейнер з інверсією управління (IoC).

Метод ConfigureServices:

- Необов'язковий параметр.
- Викликається вузлом перед методом Configure для настройки служб додатку.

- За угодою використовується для задання параметрів конфігурації.

Вузол може налаштовувати деякі служби перед викликом методів Startup.

Для функцій, які потребують значної налаштуванні, існують методи розширення Add {Service} в IServiceCollection. Наприклад, AddDbContext, AddDefaultIdentity, AddEntityFrameworkStores (Лістинг 3.4.3):

```
public void ConfigureServices(IServiceCollection services, IConfiguration
configuration)
    {
      //compression
      services.AddResponseCompression();
      //add options feature
      services.AddOptions();
      //add Easy caching
      services.AddEasyCaching();
      //add distributed memory cache
      services.AddDistributedMemoryCache();
      //add HTTP session state feature
      services.AddHttpSession();
      //add default HTTP clients
      services.AddHttpClients();
      //add anti-forgery
      services.AddAntiForgery();
      //add localization
      services.AddLocalization();
      //add theme support
      services.AddThemes();
    }
```

Лістинг 3.4.3 - Підключення сервісів додатку в методі

Додавання служб в контейнер служб робить їх доступними в додатку і в методі Configure. Служби вирішуються за допомогою впровадження залежностей або з ApplicationServices.

Конвеєр обробки запитів складається з ряду компонентів ПО проміжного шару. Кожен компонент виконує асинхронні операції в HttpContext, а потім або викликає наступний компонент в конвеєрі, або завершує запит.

Метод Configure використовується для вказівки того, як додаток реагує на HTTP-запити. Конвеєр запитів налаштовується шляхом додавання компонентів ПО проміжного шару в екземпляр IApplicationBuilder. IApplicationBuilder доступний для методу Configure, але він не зареєстрований в контейнері служби. При розміщенні створюється IApplicationBuilder і передається безпосередньо в Configure.

Шаблони ASP.NET Core налаштовують конвеєр з підтримкою наступних компонентів і функцій:

- Сторінка з відомостями про виключення для розробника
- обробників винятків;
- HTTP Strict Transport Security (HSTS)
- перенаправлення HTTPS;
- статичні файли
- ASP.NET Core MVC і Razor Pages.

За прийнятим угодою компонент ПО проміжного шару додається в конвеєр викликом методу розширення Use ... в методі Startup.Configure. Наприклад, щоб включити відображення статичних файлів, викличте UseStaticFiles.

Код настройки конвеєра обробки запитів виділений в наступному прикладі (Лістинг 3.4.4):

```
public void Configure(IApplicationBuilder application)
    {
      //use response compression
      application.UseResponseCompression();
      //use static files feature
      application.UseStaticFiles();
      //check whether requested page is keep alive page
      application.UseKeepAlive();
```

```
            //check whether database is installed
            application.UseInstallUrl();
            //use HTTP session
            application.UseSession();
            //use request localization
            application.UseRequestLocalization();
            //easy caching
            application.UseEasyCaching();
}
```

Лістинг 3.4.4 - Налаштування конвеєру обробки запитів

ASP.NET Core включає великий набір вбудованого ПО проміжного шару і дозволяє вам писати своє власне.

Кожен метод розширення Use додає один або кілька компонентів ПО проміжного шару в конвеєр запитів. Наприклад, UseStaticFiles налаштовує ПО проміжного шару для обслуговування статичних файлів.

Кожен компонент ПО проміжного шару в конвеєрі запитів відповідає за виклик наступного компонента в конвеєрі або замикає ланцюжок, якщо це необхідно.

У сигнатурі методу Configure можуть бути вказані додаткові служби, такі як IWebHostEnvironment, ILoggerFactory або будь-які служби, визначені в ConfigureServices. Ці служби впроваджуються, якщо доступні.

Для зручнішої конфігурації додатку та представлення класів, конфігурація сервісів була перенесена в окремий метод розширення ConfigureApplicationServices класу ServiceCollectionExtensions(Лістинг 3.4.5):

```
public static IServiceProvider ConfigureApplicationServices(this
IServiceCollection services,
            IConfiguration configuration, IHostingEnvironment hostingEnvironment)
        {
            //більшість API постачальників вимагає TLS 1.2 на сьогодні
            ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            //додавання параметрів конфігурації
            var Config =
services.ConfigureStartupConfig<Config>(configuration.GetSection("FiberDock")
);
```

```
        //додавання параметрів хостінгу
services.ConfigureStartupConfig<HostingConfig>(configuration.GetSection("Host
ing"));
        //додавання аксесора to HttpContext (доступ в інших частинах додатку)
        services.AddHttpContextAccessor();
        //створення дефолтного провайдера файлів
        CommonHelper.DefaultFileProvider = new
FileProvider(hostingEnvironment);
        //ініціалізація плагінів
        var mvcCoreBuilder = services.AddMvcCore();
        mvcCoreBuilder.PartManager.InitializePlugins(Config);
        //create engine and configure service provider
        var engine = EngineContext.Create();
        var serviceProvider = engine.ConfigureServices(services, configuration,
Config);
        // подальші дії виконуються лише тоді, коли база даних встановлена
        if (!DataSettingsManager.DatabaseIsInstalled)
           return serviceProvider;
        //ініціалізація та старт сервісу завдань
        TaskManager.Instance.Initialize();
        TaskManager.Instance.Start();
        //старт додатку логування
        engine.Resolve<ILogger>().Information("Application started");
        //встановлення плагінів додатку
        engine.Resolve<IPluginService>().InstallPlugins();
        return serviceProvider;
}
```

Лістинг 3.4.5 - Реалізація методу розширення конфігурації сервісів додатку

Для більш детального показу функціоналу сервісів, у лістингу 3.4.6 представлено реалізацію сервісу AddAntiForgery та AddHttpSession, в яких представлено налаштування параметрів даних сервісів для контролю cookie підключення:

```
public static void AddAntiForgery(this IServiceCollection services)
    {
        //перезапис імені cookie
```

```
            services.AddAntiforgery(options =>
            {
                options.Cookie.Name =
$"{CookieDefaults.Prefix}{CookieDefaults.AntiforgeryCookie}";
                // чи дозволяти використовувати файли cookie з підробки на
захищеній SSL сторінці на інших сторінках магазину, в данному випадку
заборонено
                options.Cookie.SecurePolicy =
DataSettingsManager.DatabaseIsInstalled &&
EngineContext.Current.Resolve<SecuritySettings>().ForceSslForAllPages
                    ? CookieSecurePolicy.SameAsRequest : CookieSecurePolicy.None;
            });
        }
        public static void AddHttpSession(this IServiceCollection services)
        {
            services.AddSession(options =>
            {
                options.Cookie.Name =
$"{CookieDefaults.Prefix}{CookieDefaults.SessionCookie}";
                options.Cookie.HttpOnly = true;
                // чи дозволяти використання значень сеансу із захищеної SSL
сторінки на інших сторінках магазину, в данному випадку не дозволено
                options.Cookie.SecurePolicy =
DataSettingsManager.DatabaseIsInstalled &&
EngineContext.Current.Resolve<SecuritySettings>().ForceSslForAllPages
                    ? CookieSecurePolicy.SameAsRequest : CookieSecurePolicy.None;
            });
}
```

Лістинг 3.4.6 - Реалізація та передача модифікації

ASP.NET Core має вбудовану підтримку аутентифікації на основі кукі. Для цього в ASP.NET визначений спеціальний компонент middleware, який серіалізуются дані користувача в зашифровані аутентифікаційні куки і передає їх на сторону клієнта. При отриманні запиту від клієнта, в якому містяться аутентифікаційні кукі, відбувається їх валідація, десеріалізація і ініціалізація властивості User об'єкта HttpContext.

Для установки аутентифікації за допомогою кукі в виклик services.AddAuthentication передається значення CookieAuthenticationDefaults. AuthenticationScheme. Далі за допомогою методу AddCookie () налаштовується аутентифікація. По суті в цьому методі проводиться конфігурація об'єкта CookieAuthenticationOptions, який описує параметри аутентифікації за допомогою кук. Зокрема, в даному випадку використано одну властивість CookieAuthenticationOptions - LoginPath. Це властивість встановлює відносний шлях, по якому буде перенаправлятися анонімний користувач при доступі до ресурсів, для яких потрібна аутентифікація.

В лістингу 3.4.7 представлена реалізація методу розширення для додавання сервісу аутентифікації за логіном\паролем, кукі, зовнішніми плагінами (Google Account та ін).

```
public static void AddAuthentication(this IServiceCollection services)
    {
        //встановити дефолтні схеми аутентифікації
        var authenticationBuilder = services.AddAuthentication(options =>
        {
            options.DefaultChallengeScheme =
AuthenticationDefaults.AuthenticationScheme;
            options.DefaultScheme =
AuthenticationDefaults.AuthenticationScheme;
            options.DefaultSignInScheme =
AuthenticationDefaults.ExternalAuthenticationScheme;
        });
        //Додавання аутентифікації по кукі
authenticationBuilder.AddCookie(AuthenticationDefaults.AuthenticationScheme,
options =>
        {
            options.Cookie.Name =
$"{CookieDefaults.Prefix}{CookieDefaults.AuthenticationCookie}";
            options.Cookie.HttpOnly = true;
            options.LoginPath = AuthenticationDefaults.LoginPath;
            options.AccessDeniedPath = AuthenticationDefaults.AccessDeniedPath;
            // чи дозволяти використовувати файли cookie аутентифікації із
захищеної SSL сторінки на інших сторінках магазину (заборонити)
```

```
            options.Cookie.SecurePolicy =
DataSettingsManager.DatabaseIsInstalled &&
EngineContext.Current.Resolve<SecuritySettings>().ForceSslForAllPages
                ? CookieSecurePolicy.SameAsRequest : CookieSecurePolicy.None;
        });
        //додавання аутентифікації з інших сорсів
authenticationBuilder.AddCookie(AuthenticationDefaults.ExternalAuthenticationS
cheme, options =>
        {  options.Cookie.Name =
$"{CookieDefaults.Prefix}{CookieDefaults.ExternalAuthenticationCookie}";
            options.Cookie.HttpOnly = true;
            options.LoginPath = AuthenticationDefaults.LoginPath;
            options.AccessDeniedPath = AuthenticationDefaults.AccessDeniedPath;
            options.Cookie.SecurePolicy =
DataSettingsManager.DatabaseIsInstalled &&
EngineContext.Current.Resolve<SecuritySettings>().ForceSslForAllPages
                ? CookieSecurePolicy.SameAsRequest : CookieSecurePolicy.None;
        });
        // реєстрація та налаштування плагінів зовнішньої аутентифікації
        var typeFinder = new WebAppTypeFinder();
        var externalAuthConfigurations =
typeFinder.FindClassesOfType<IExternalAuthenticationRegistrar>();
        var externalAuthInstances = externalAuthConfigurations
            .Select(x =>
(IExternalAuthenticationRegistrar)Activator.CreateInstance(x));

        foreach (var instance in externalAuthInstances)
            instance.Configure(authenticationBuilder);
}
```

Лістинг 3.4.7 - Реалізація методу розширення сервісу аутентифікації

Загалом, реалізація класів у рамках фреймворку ASP.NET Core є реалізацією механік доступу та відображення даних на сервері.

4 ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Розгортання програмної системи та системні вимоги

Загалом при розгортанні додатка ASP.NET Core в середовищі зовнішнього розміщення виконуються наступні дії.

- Розгортання опублікованої програми у папці на сервері розгортання.
- Налаштування диспетчера процесів, який запускає додаток при надходженні запитів і перезапускає його після аварійного завершення або після перезавантаження сервера.
- Для конфігурації налаштуйте такий проксі-сервер, який перенаправляє запити в додаток.

Команда інтерфейсу командного рядка dotnet publish компілює код програми та копіює файли, необхідні для його виконання, в папку publish[23]. При розгортанні з Visual Studio крок dotnet publish виконується автоматично перед копіюванням файлів до місця розгортання.

Папка publish містить один або кілька файлів збірки і, залежно від додатку, також може включати середу виконання .NET.

Додатки .NET Core можуть публікуватися як автономні розгортання або розгортання, що залежать від платформи. Якщо додаток автономний, в папку publish додаються файли збірки, що містять середу виконання .NET[24]. Якщо додаток залежить від платформи, операційне середовище програмного .NET не повинне додаватися, так як додаток посилається на версію .NET, встановлену на сервері. За замовчуванням використовується модель розгортання із залежністю від платформи.

На додаток до EXE- і DLL-файлів папка публікації для додатка ASP.NET Core зазвичай містить файли конфігурації, статичні ресурси та представлення MVC[25].

Додаток ASP.NET Core - це консольний додаток, який повинен запускатися при завантаженні сервера і перезапускати після його аварійного

завершення. Для автоматичного запуску і перезапуску потрібно диспетчер процесів. Далі наведені найбільш поширені диспетчери процесів для ASP.NET Core:

- Linux
    - Nginx
    - Apache
- Windows
    - служби IIS
    - служба Windows

Якщо додаток використовує сервер Kestrel, ви можете використовувати Nginx, Apache або IIS в якості зворотного проксі-сервера. Зворотний проксі-сервер отримує HTTP-запити з Інтернету і пересилає їх в Kestrel.

Будь-яка з цих конфігурацій, зі зворотним проксі-сервером чи без нього, є підтримуваною конфігурацією розміщення.

4.1.1 Використання серверу Kestrel

Kestrel - це кросплатформенний веб-сервер для ASP.NET Core. Веб-сервер Kestrel за замовчуванням включається в шаблони проектів ASP.NET Core.

Протокол HTTP/2 доступний для додатків ASP.NET Core, якщо виконані наступні базові вимоги:

- Операційна система:
    - Windows Server 2016 / Windows 10 або більш пізніх версій;
    - Linux з OpenSSL 1.0.2 або пізнішої версії (наприклад, Ubuntu 16.04 або більш пізньої версії).
- Необхідна версія .NET Framework: .NET Core версії 2.2 або більш пізня версія
- Підключення з підтримкою узгодження протоколу рівня додатків (ALPN).

- Підключення TLS 1.2 або більш пізньої версії.

Kestrel можна використовувати окремо або зі зворотним проксі-сервером, таким як IIS, Nginx або Apache. Зворотний проксі-сервер отримує HTTP-запити з мережі і пересилає їх в Kestrel[26].

Kestrel використовується в якості веб-сервера переходу (з виходом в Інтернет)(Рисунок 4.1.1.1).

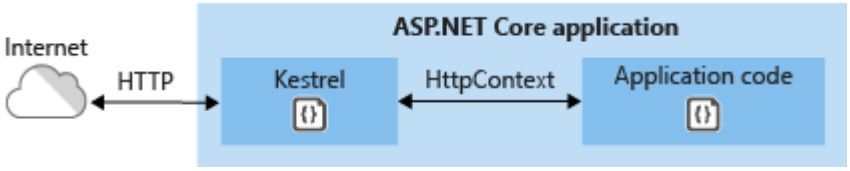


Рисунок 4.1.1.1 - Схема виходу в Інтернет без зворотного проксі-сервера

Kestrel використовується в конфігурації зворотного проксі-сервера (Рисунок 4.1.1.2).

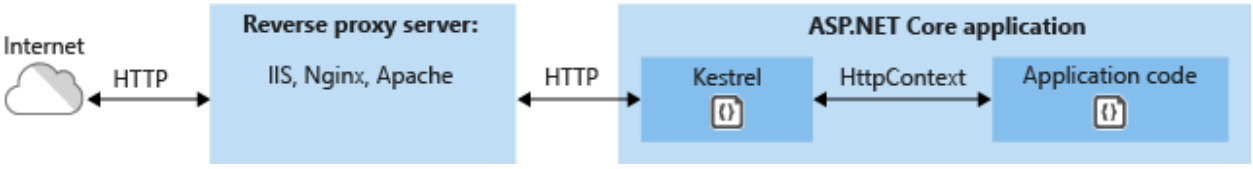


Рисунок 4.1.1.2 - Kestrel взаємодіє з Інтернетом побічно, через зворотний проксі-сервер, такий як IIS, Nginx або Apache.

Будь-яка з цих конфігурацій, зі зворотним проксі-сервером і без нього, є підтримуваної конфігурацією для розміщення.

Kestrel, який використовується в якості крайнього сервера без зворотного проксі-сервера, не підтримує загальний доступ декількох процесів до одних і тих же IP-адресів і портів. Коли Kestrel налаштований на очікування передачі даних від порту, Kestrel обробляє весь трафік для цього порту незалежно від заголовка Host запитів. Тому зворотний проксі-сервер, який може спільно використовувати порти, здатний пересилати запити в Kestrel з унікальною IP-адресою і портом.

Навіть якщо зворотний проксі-сервер не потрібно, його використання може виявитися зручним.

Зворотний проксі-сервер:

- Може обмежити загальнодоступну контактну зону розміщених на ньому програм.
- Надає додатковий рівень конфігурації і захисту.
- Може краще інтегруватися з існуючою інфраструктурою.
- Спрощує налаштування балансування навантаження і безпечних підключень (HTTPS). Сертифікат X.509 потрібен тільки для зворотного проксі-сервера, а сам цей сервер може обмінюватися даними з серверами додатку у внутрішній мережі за звичайним протоколу HTTP.

4.1.2 Розгортання додатку на веб-сервері IIS

При використанні IIS або IIS Express додаток запускається одним із таких способів:

- У тому ж процесі, що і робочий процес IIS (модель внутріпроцесного розміщення), з використанням HTTP-сервера IIS. Внутріпроцессное розміщення є рекомендованою конфігурацією.
- В процесі окремо від робочого процесу IIS (модель позапроцесного розміщення) з використанням сервера Kestrel.

Модуль ASP.NET Core - це власний модуль IIS, який обробляє власні запити IIS між службами IIS і HTTP-сервером IIS (внутріпроцесно) або сервером Kestrel.

Так як додатки ASP.NET Core виконуються в процесі, відділеному від робочого процесу IIS, модуль ASP.NET Core керує управлінням процесами. Модуль запускає процес для додатка ASP.NET Core при надходженні першого запиту і перезапускає додаток при збої або завершенні роботи. Це, по суті, збігається з поведінкою додатків, які виконуються внутріпроцесно і керованих службою активації процесів Windows (WAS).

На наступній схемі (Рисунок 4.1.2.1) показано зв'язок між IIS, модулем ASP.NET Core і додатком, розміщеним поза процесом.

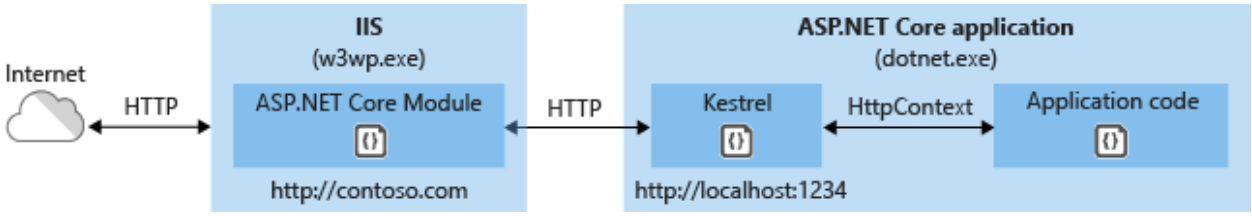


Рисунок 4.1.2.1 - Модуль ASP.NET Core в сценарії позапроцесного розміщення

Запити надходять з Інтернету в драйвер HTTP.sys в режимі ядра. Драйвер направляє запити до служб IIS на налаштований порт веб-сайту - зазвичай 80 (HTTP) або 443 (HTTPS). Модуль перенаправляє запити Kestrel на випадковий порт для додатка, що відрізняється від порту 80 або 443.

Модуль задає порт за допомогою змінної середовища під час запуску, а розширення UseIISIntegration налаштовує сервер для прослуховування http: // localhost: {PORT}. Виконуються додаткові перевірки, та інші запити не з модуля відхиляються. Модуль не підтримує переадресацію по HTTPS, тому запити переадресовуються по протоколу HTTP, навіть якщо були отримані IIS по протоколу HTTPS.

Після того як Kestrel забирає запит з модуля, запит передається в конвеєр ПО проміжного шару ASP.NET Core. Конвеєр ПО проміжного шару обробляє запит і передає його в якості примірника HttpContext в логіку програми. ПО проміжного шару, доданий інтеграцією IIS, оновлює схему, віддалений IP-адреса і базовий шлях для переадресації запиту в Kestrel. Відгук додатки передається назад в службу IIS, яка відправляє його назад в HTTP-клієнт, який ініціював запит.

Конфігурація сервера IIS для персональних комп'ютерів відбуваєтья у кілька кроків:

1. Послідовно виберіть Панель керування - Програми - Програми та засоби - Включення або відключення компонентів Windows (в лівій частині екрана).

2. Розгорніть вузол Служби IIS. Розгорніть вузол Засоби управління веб-сайтом.
3. Встановіть прапорець Консоль управління IIS.
4. Встановіть прапорець Служби Інтернету[37].
5. У групі Служби Інтернету залиште компоненти за замовчуванням або налаштуйте компоненти IIS (Рисунок 4.1.2.2).
6. Якщо під час установки IIS потрібно перезавантаження, перезавантажте систему.

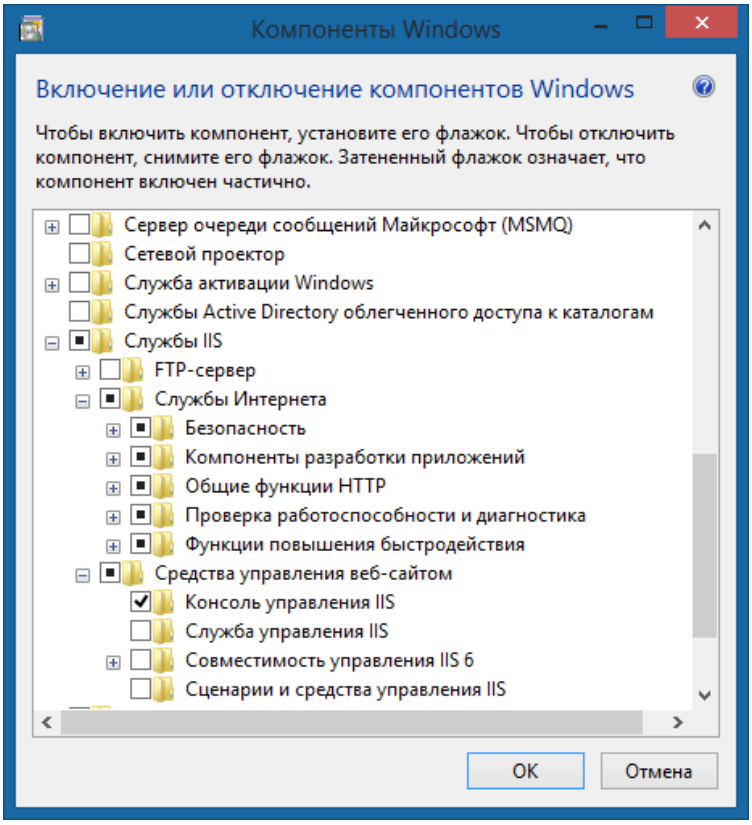


Рисунок 4.1.2.2 - Результат увімкнення служби IIS

7. Встановіть пакет розміщення .NET Core в системі розміщення. У складі пакету встановлюються область виконання .NET Core, бібліотека .NET Core і модуль ASP.NET Core. Модуль дозволяє запускати додатки ASP.NET Core під керуванням IIS.

Процес розміщення виконавчих файлів в root папці серверу можна описати у кілька кроків таким чином:

1. Натиснути праву кнопку миші - обрати пункт Publish (Рисунок 4.1.2.3).

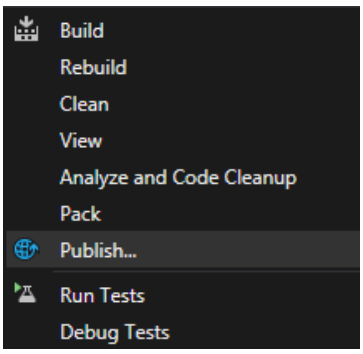


Рисунок 4.1.2.3 - Перший етап публікації файлів до root папки веб-серверу

2. У новому вікні середовища розробки натиснути Publish - Start - обрати пункт Folder[34].
3. У стрічці шляху розміщення root папки веб-сервера встановити шлях до папки, у котру буде розміщено виконавчі файли сервера (Рисунок 4.1.2.4) (Раджу обирати логічний диск, на котрому не розміщена операційна система задля уникання суперечностей з дозволами).

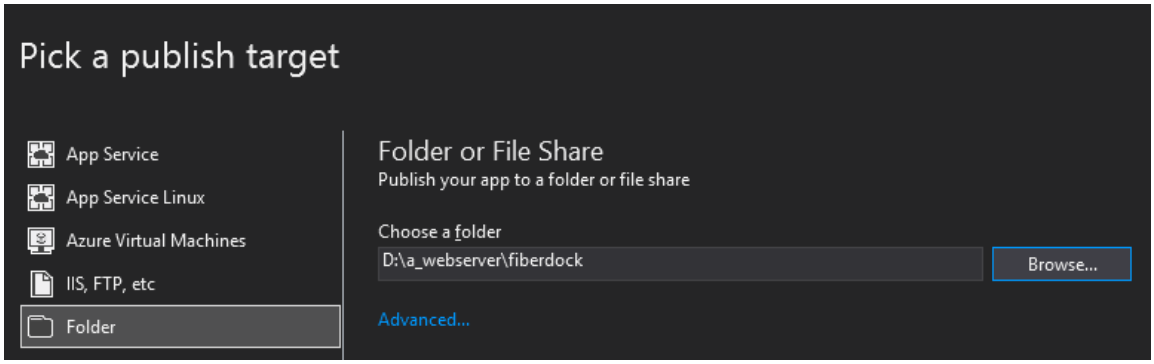


Рисунок 4.1.2.4 - Вікно вибору шляху до root папки веб-серверу

4. Натиснути кнопку Create profile, а потім натиснути кнопку Publish. Всі необхідні файли будуть скомпільовані системою та розміщені в root папці веб-серверу.

Процес запуску сайту також у свою чергу потребує кілька кроків[39]:

1. Якщо процес розміщення файлів не був виконаний, тоді в системі створіть папку, в якій будуть зберігатися файли і папки опублікованого додатка. На наступному етапі шлях до папки надається IIS як фізичний шлях до програми.

2. У вікні диспетчера IIS на панелі Підключення розгорніть вузол сервера. Клацніть правою кнопкою миші папку Сайти. У контекстному меню виберіть пункт Додати веб-сайт[37].
3. Вкажіть ім'я в поль Ім'я сайту і задайте значення в поле Фізичний шлях для створеної папки розгортання програми. Вкажіть конфігурацію прив'язки і натисніть кнопку ОК, щоб створити веб-сайт (Рисунок 4.1.2.5).

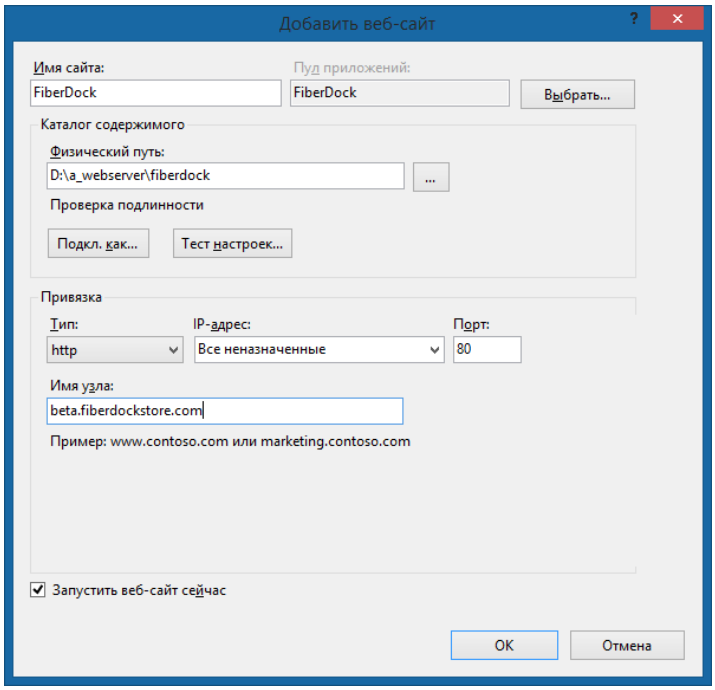


Рисунок 4.1.2.5 - Конфігурація прив'язки створюваного сайту

4. Розгорніть вузол сервера і виберіть Пули додатків.
5. Клацніть правою кнопкою миші пул додатків сайту і в контекстному меню виберіть пункт Основні параметри.
6. У вікні Зміна пулу додатків задайте для параметра Версія середовища CLR .NET значення Без керованого коду(Рисунок 4.1.2.6).

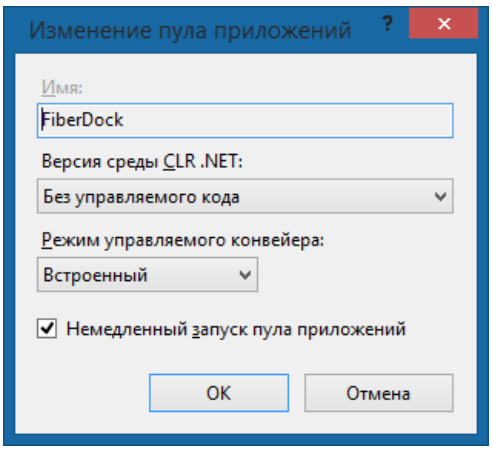


Рисунок 4.1.2.6 - Зміна версії середи CLR.NET

7. ASP.NET Core 2.2 або більш пізньої версії: для 64-розрядного (x64) автономного розгортання, в якому використовується модель розміщення в процесі, вимкніть пул додатків для 32-розрядних (x86) процесів.
8. Переконайтеся в тому, що посвідчення моделі процесу має відповідні дозволи.
9. Можна повторити цю дію на інших комп'ютерах серверах, таким чином створимо мережу сайтів з однією базою даних, в якій аутентифікація перевіряється за допомогою флагу активності в базі даних та прив'язки до кукі.

В результаті, коли ми перейдемо у браузері за адресою beta.fiberdockstore.com, ми відкриємо головну сторінку магазину(Рисунок 4.1.2.7):

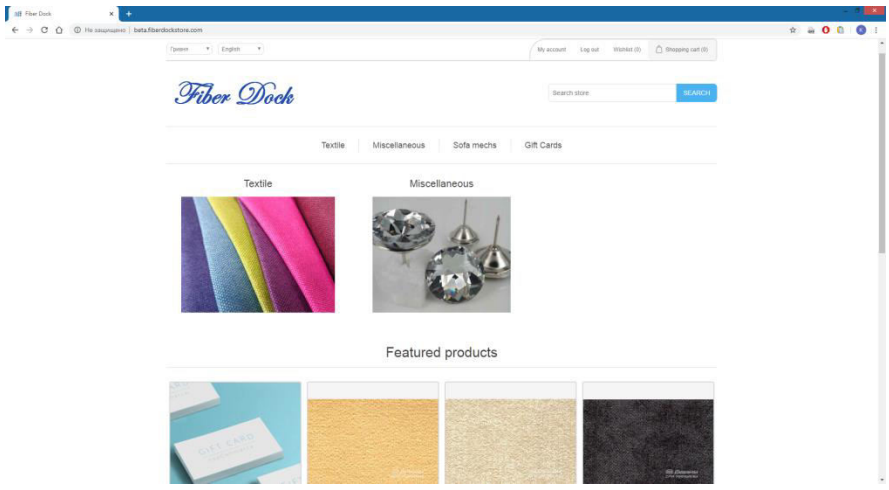


Рисунок 4.1.2.7 - Головна сторінка сайту після розгортання

4.2 Опис типових схем використання системи

Для клієнта, найтиповішою дією на сайті є відкриття самого сайту, все що необхідно - це перейти за веб-адресою beta.fiberdockstore.com і відкриється голвна сторінка сайту(Рисунок 4.2.1):

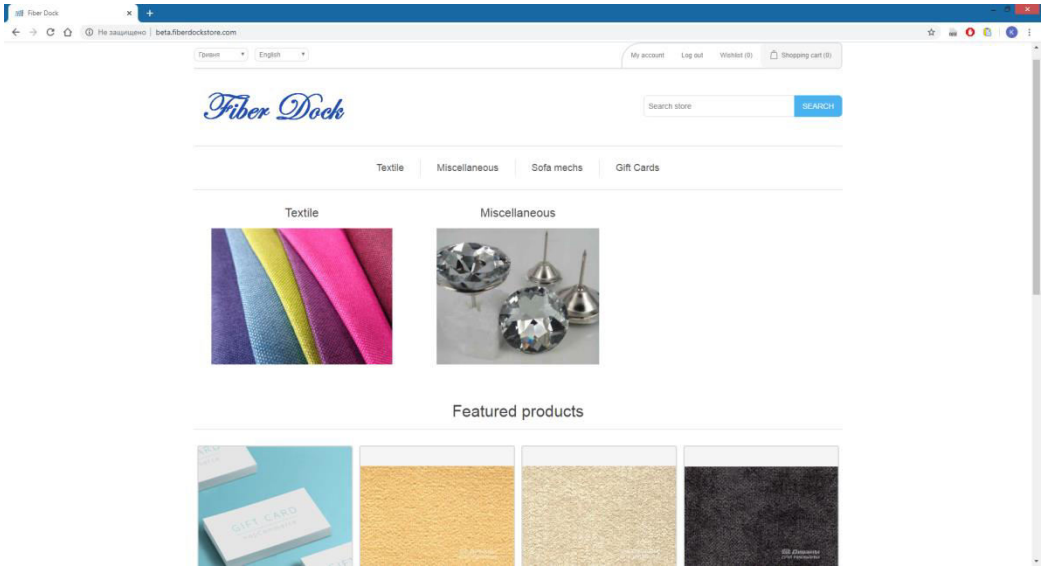


Рисунок 4.2.1 - Головна сторінка сайту

Наступною дією будь-якого користувача буде перегляд категорій в магазині товарів для ремонту меблів, наприклад користувач шукає Шеніл у категорії Тканина, користучаву буде преставлена сторінка з видами тканин (Рисунок 4.2.2):

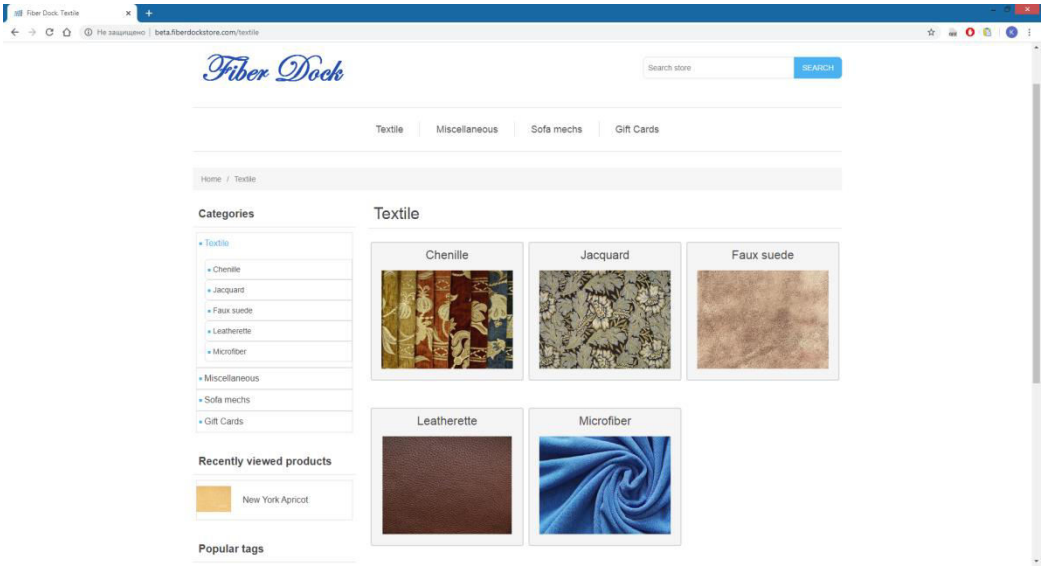


Рисунок 4.2.2 - Представлення підкатегорій у категорії Тканина

Користувач вибирає Шеніл, та з переходом на нову сторінку бачить каталог Шенілу(Рисунок 4.2.3), наявного у продажі на сайті, він може додати товар до Кошика, чи додати його до списку бажань(Рисунок 4.2.4).

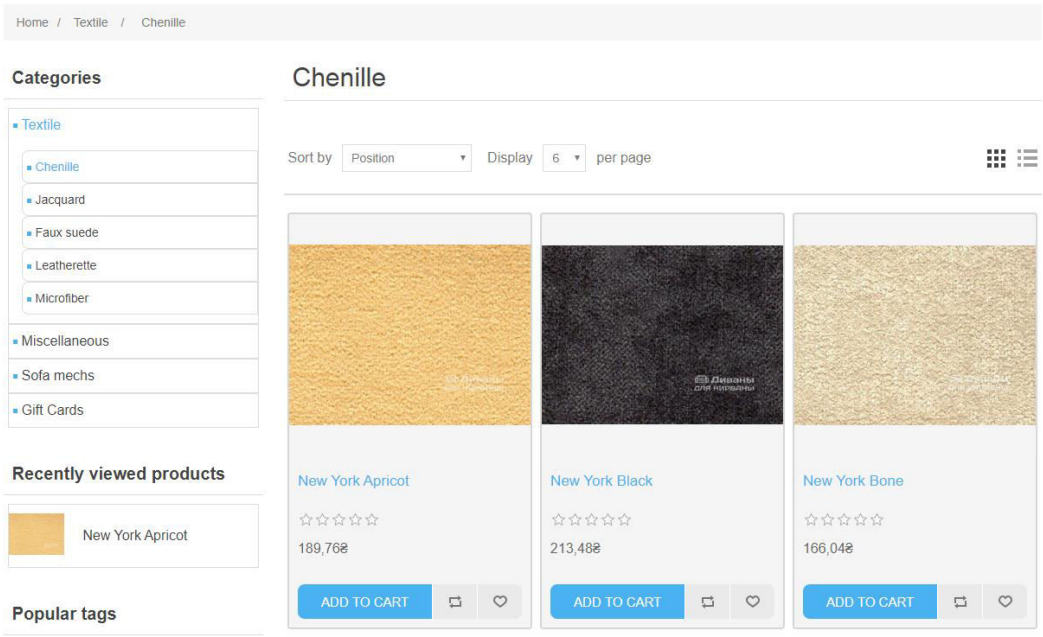


Рисунок 4.2.3 - Каталог шенілу наявного у продажі на сайті

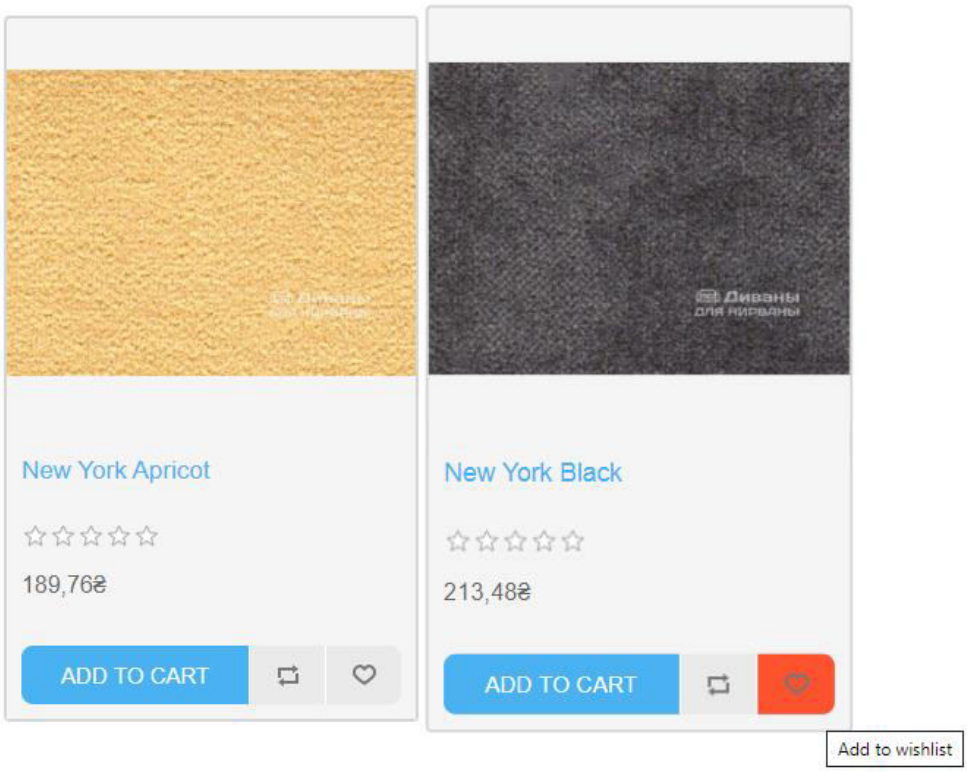


Рисунок 4.2.4 - Можливість додавання товару до кошика чи списку бажань

Користувач сайту після додавання товару до Кошика, може перейти до самого Кошика задля оформлення покупки, вказуючи кількість товару, адресу доставки , поштовий індекс та інше(Риусунок 2.4.5).

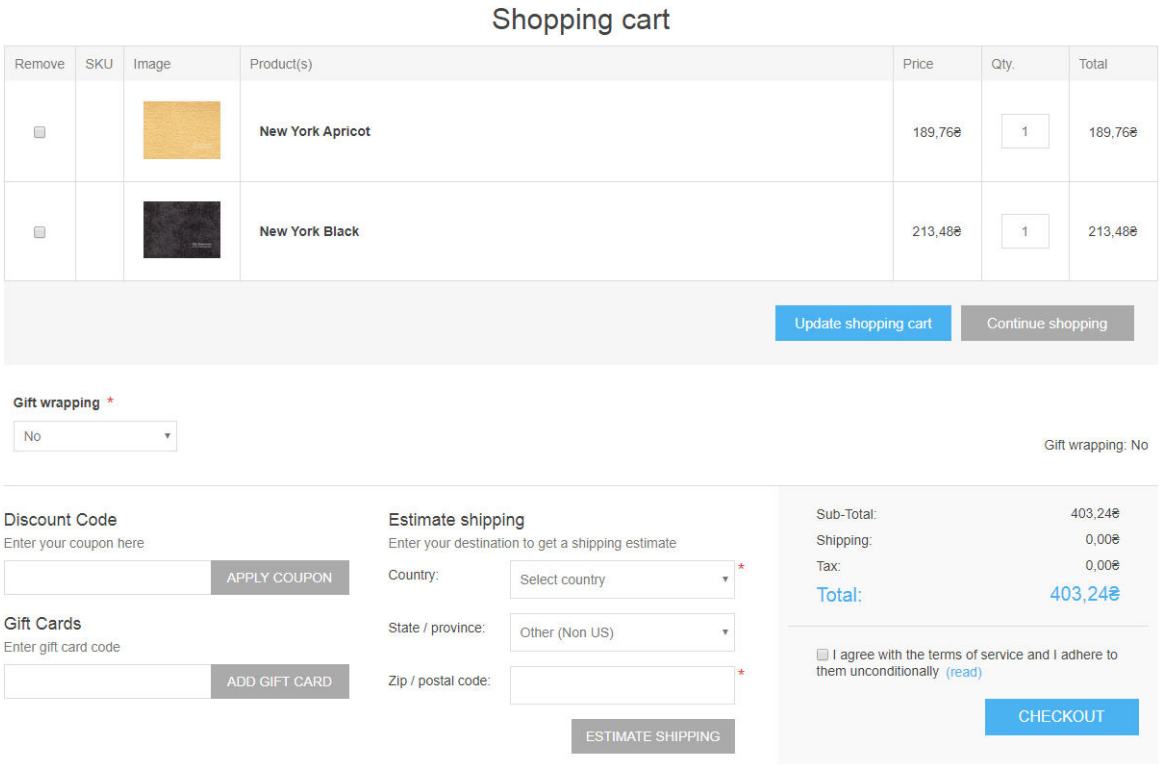


Рисунок 2.4.5 - Скріншот кошика з обраними товарами

Якщо користувач(клієнт) є постійним покупцем на сайті, йому буде вигідно створити особистий акаунт та оформлювати покупки з нього для накопичення знижки, а отже користувач може зайти у свій особистий акаунт чи зареєструватись за його відсутності(Рисунок 2.4.6).

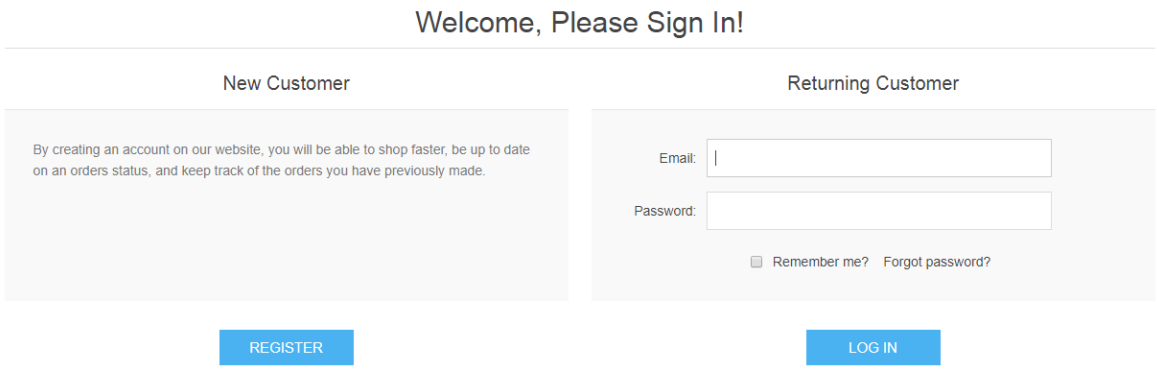


Рисунок 2.4.6 - Сторінка входу в особистий кабінет

5 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

5.1 План використання Unit-тестів

До запуску програми в виробництво, коли воно стане доступним користувачам, важливо переконатися, що дане додаток функціонує, як і повинно, що в ньому немає помилок. Для перевірки додатку ми можемо використовувати різні схеми і механізми тестування. Одним з таких механізмів є юніт-тести.

Юніт-тести дозволяють швидко і автоматично протестувати окремі компоненти програми незалежно від іншої його частини. Не завжди юніт-тести можуть покрити весь код програми, але тим не менше вони дозволяють істотно зменшити кількість помилок вже на етапі розробки.

Ми не повинні тестувати код використовуваного фреймворка або використовуваних залежностей. Тестувати треба тільки той код, який написали ми самі.

Треба відзначити, що в цілому концепція юніт-тестів не є непорушною вимогою до веб-розробці, та й взагалі до розробки. Хтось вважає, що юніт-тести обов'язково повинні покривати весь код проекту, хтось вважає, що юніт-тести можна використовувати переважно для особливо складних моментів у коді програми, якоїсь складної логіки. Деякі не використовують юніт-тести.

Але тим не менше юніт-тести несуть потенційні переваги при розробці, до яких слід віднести не тільки власне перевірку результату і тестування коду, але і інші, як наприклад, написання слабосвязанних компонентів відповідно до принципів SOLID. Адже щоб тестувати компоненти програми незалежно один від одного, нам треба, щоб вони були слабо зв'язаної. А подібне побудова додатки в подальшому може позитивно позначитися на його подальшій модифікації і підтримки.

Більшість юніт-тестів так чи інакше мають ряд таких ознак:

Тестування невеликих ділянок коду ("юнітів")

Для створення юніт-тестів вибираються невеликі ділянки коду, які треба протестувати. Тестований ділянку, як правило, повинен бути менше класу. У більшості випадків тестується окремий метод класу або навіть частина функціоналу методу. Упор на невеликі ділянки дозволяє досить швидко писати простенькі тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в юніт-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

Створення юніт-тестів для невеликих ділянок коду веде до того, що кількість цих юніт-тестів стає дуже велике. Якщо процес отримання результатів і проведення тестів не автоматизовано, то це може привести до непродуктивної витраті робочого часу і зниження продуктивності. Тому важливо, щоб результати юніт-тестів представляли собою просте рішення, яке означає, пройдений тест чи ні. Для автоматизації процесу розробники зазвичай звертаються до фреймворками юніт-тестування.

Навіть невеликі зміни в класі можуть привести до невдачі багатьох юніт-тестів, оскільки реалізація використовуваного класу змінилася. Тому при написанні юніт-тестів обмежуються тільки загальнодоступними кінцевими точками, що дозволяє ізолювати юніт-тести від багатьох деталей внутрішньої реалізації компонента. В результаті зменшується ймовірність, що зміни в класах можуть привести до провалу юніт-тестів.

5.2 Розробка Unit-тестів

Порядок написання коду при TDD досить простий:

1. Пишемо юніт-тест
2. Запускаємо його і бачимо, що він завершився невдачею (програмний код адже ще не написаний)
3. Пишемо кілька коду, достатню для запуску тесту
4. Знову запускаємо тест і бачимо його результати

Цей цикл повторюється знову і знову, поки не буде закінчено роботу над програмним кодом. Так як більшість фреймворків юніт-тестування позначають невдалі тести червоним кольору (наприклад, виводиться текст червоного кольору), а вдалий тест відзначається зеленим кольором (знову ж виводиться текст зеленого кольору), то даний цикл часто називають червоним / зеленим циклом.

Налаштування модульних тестів для дій контролера, для зосередження на поведінці контролера. У модульних тестах контролера не враховуються такі аспекти, як фільтрація, маршрутизація і (або) прив'язка моделі. Для прикладу в лістингу 5.2.1 наведена реалізація тестів перевірки паролей під час їх зміни користувачем.

```
public class ChangePasswordValidatorTests : BaseValidatorTests
   {
      private ChangePasswordValidator _validator;
      private CustomerSettings _customerSettings;
      [SetUp]
      public new void Setup() {
         _customerSettings = new CustomerSettings();
         _validator = new ChangePasswordValidator(_localizationService,
_customerSettings);}
      [Test]
      public void Should_have_error_when_oldPassword_is_null_or_empty()
      {
        var model = new ChangePasswordModel{OldPassword = null};
         _validator.ShouldHaveValidationErrorFor(x => x.OldPassword, model);
         model.OldPassword = "";
         _validator.ShouldHaveValidationErrorFor(x => x.OldPassword, model);
```

```
[Test]
public void Should_not_have_error_when_oldPassword_is_specified()
{
    var model = new ChangePasswordModel{
    OldPassword = "old password"
 };
    _validator.ShouldNotHaveValidationErrorFor(x => x.OldPassword,
model); }
[Test]
public void Should_have_error_when_newPassword_is_null_or_empty()
{
  var model = new ChangePasswordModel{NewPassword = null};
    //we know that new password should equal confirmation password
    model.ConfirmNewPassword = model.NewPassword;
    _validator.ShouldHaveValidationErrorFor(x => x.NewPassword, model);
    model.NewPassword = "";
    model.ConfirmNewPassword = model.NewPassword;
    _validator.ShouldHaveValidationErrorFor(x => x.NewPassword, model); }
[Test]
public void Should_not_have_error_when_newPassword_is_specified(){
    var model = new ChangePasswordModel{NewPassword = "new
password"};
    model.ConfirmNewPassword = model.NewPassword;
    _validator.ShouldNotHaveValidationErrorFor(x => x.NewPassword,
model); }
```

Лістинг 5.2.1 - Тест перевірки зміни паролю

Якщо створюються користувацькі фільтри і маршрути, проводити для них модульне тестування слід ізольовано, але не в рамках тестування певної дії контролера.

6 ОРГАНІЗАЦІЙНО–ЕКОНОМІЧНА ЧАСТИНА

6.1 Загальний підхід до визначення економічної ефективності розробки

Обов’язковою складовою частиною виконуваного проекту дипломної роботи за темою "Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core" є визначення фінансових витрат на різних етапах виконання робіт. Відповідно, важливо вірно здійснити фінансову оцінку передбачуваних витрат, продуктивність, корисність та, в результаті, економічну ефективність проекту.

Наукоємні розробки та дослідження, на відміну від корпоративних, не завжди супроводжують за мету отримання прибутку або ж іншої матеріальної вигоди. В багатьох випадках, проекти наукового спрямування не є економічно вигідними. Однак, вони є рушійною силою прогресу, дослідженням незвіданих галузей та проблем, які, у свою чергу, майже завжди впливають на майбутні різнопланові розробки. Тому дуже часто наукова діяльність стимулюється зовнішніми інвестиціями та підтримкою держаних інститутів, міжнародних грантів. В плані використання результатів досліджень та на основі отриманих моделей можна робити прогнози по впровадженню нових методик та принципів організації роботи діагностичних установ. Різноманітні медичні центри зможуть скористатися на практиці розробленою системою для покращення існуючих та отримання нових діагностичних методик.

Оцінка вартості дослідницьких розробок базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації. Так, як результати роботи у вигляді математичних моделей та реалізованого на їх основі ПЗ не буде використовуватися в комерційних цілях та не підлягатиме продажу, а становить наукову та

інтелектуальну цінність, то доходу від продажі ПЗ та розробки як такого не передбачається. Іншими словами, всі вкладені кошти та витрати на розробку даного рішення є не взаємоокупними, що несуть лише витрати у кількості залучених ресурсів та матеріальних засобів.

Згідно Статті 8 Закону № 3792-12 передбачено, що твори наукового характеру та комп'ютерні програми є об'єктами авторського права [28]. У разі реєстрації виконаної роботи як інтелектуальної власності та авторського права у державній службі інтелектуальної власності України необхідно буде сплатити збори за державну реєстрацію (382,5 грн.).

Для отримання відмінних результатів експериментів та доцільності розробки такого спеціалізованого ПЗ потрібні відповідні затрати на дослідження та розробку. Це і становитиме основу витрат, які будуть здійснені протягом підготовки та виконання реалізації даного рішення.

Уявно модель витрат можна поділити на дві основні частини: витрати, пов'язанані на дослідження предметної області, побудову математичних моделей, отримання попередніх результатів експериментів, та частину реалізації програмної системи, архітектури та тестування.

Враховуючи залежність якості кінцевого продукту від кваліфікації програмістів, потрібно сконцентрувати увагу на якості та результативності розробки. Для цього потрібно провести ґрунтовний аналіз предметної області, залучити найновіші та інноваційні технології, провести ґрунтовне тестування та оцінку результатів розробки. Для забезпечення хорошої результативності при розробці доводиться йти на додаткові заходи заохочення та стимулювання у вигляді преміювання працівників, підтримання наукового дослідження досвідом іноземних науковців, дорогоцінних лабораторних дослідів.

До створення ПЗ можуть бути залучені позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох випадках співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором

підряду, то все залежить від того, чи зареєстрований виконавець підприємцем. Важливим етапом розробки ПЗ є його тестування, яке виконується тестувальником за певну винагороду. Якщо тестувальники не перебувають з підприємством у трудових відносинах, оплата виконується на основі договору підряду на виконання робіт з тестування ПЗ. Сам же результат розробки не оподатковується, адже не є комерційним проектом і не спрямований на продаж.

6.2 Розрахунок вартості процесу розробки та оцінка економічної ефективності проекту

Для оцінки нематеріальних активів використовують міжнародні стандарти оцінки розрахунку вартості об’єктів інтелектуальної власності, розроблені TIAVIS (The International Assets Vabulation Standarts Commitie).

Виконання розробки програмного забезпечення з огляду економічної моделі можна виконувати двома способами: процедурним та об’єктно-орієнтованим. Обидва підходи потребують залучення ресурсів у вигляді програмісті-розробників, тестувальників, керівника проекту, наукового ресурсу. Різниця виникає в самій схемі розробки, тривалості періоду розробки та відповідній вартості. Процедурний підхід для розробки ПЗ в основі якого лежать процедури і функції передбачає розробку ПЗ як монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію. Об’єктно-орієнтований підхід, що ґрунтується на основі об’єктів певних класів, що описують певну область, описують певну поведінку (методи) та володіють властивостями (атрибутами), орієнтовані на варіанти використання та покроковий процес розробки[28].

Для початку робіт необхідно скласти технічне завдання на розробку, яке є основним документом, що регламентує подальшу роботу, та містить докладний опис необхідних функцій програми, інтерфейс, технології, інше.

Вартість складання технічного завдання переважно складає до 10% від планованої вартості розробки. Роботу зі складання технічного завдання веде керівник проекту разом із програмістами та консультуючись із замовником.

Усі програмісти, що працюють у штаті підприємства-розробника мають встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлено у таблиці 6.2.1. Всі суми наведені в національній валюті – в гривні.

Таблиця 6.2.1 – Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Фронт-енд розробник	13860,00	630,00	21	13230,00	28	17540,00
Програміст паралельного ПЗ	12540,00	570,00	21	11970,00	28	15960 ,00
Тестувальник	9750,00	390,00	7	2730,00	7	2730,00
Знавець ОО	9790,00	445,00	10	4450,00	10	4450,00
Додаткова зар. плата 20%			38	6476,00	40	8136,00
Фонд оплати праці 36,77%				11906,126		13696,96
Всього витрат на зар. плату				50726,13		62512,96
Військовий збір 1,5%				761,43		937,69
Єдиний соціальний внесок 3.6%				1826,14		2250,47

ПДВ, 15%	7608,92	9376,94
Всього	**60922,62**	**75078,06**

Згідно вимог та прорахованої кількості необхідних ресурсів на виконання, розробку, тестування та дослідницьку роботу було отримано основні часові рамки роботи над проектом. Так для об’єктно-орієнтованого підходу загальна тривалість роботи над ПЗ становить 38 робочих днів (під робочим днем розуміється 8-ми годинний робочий день), що включає роботу програміста паралельного ПЗ, який, в свою чергу, являється і керівником розробки, роботу тестувальника, фронт-енд розробника та знавця об'єктної області(ОО). Сума витрат на заробітню плату становить 60922,62 гривень включаючи всі види додаткових оплат. Для процедурного підходу до розробки суми дещо більші, адже затрачається більше часу на розробку. Так, при використанні процедурного підходу сумарна тривалість часу розробки становить 40 робочі дні, та витрати у вигляді виплат заробітної плати становлять 75078,06 гривень.

Витрати на науково-дослідницьку роботу та здійснення розробки програмних продуктів і об’єктно-орієнтованим, і процедурним способом включають:

Основна заробітна плата:

$ЗП_{осн\ 1} = 32380$ грн; $ЗП_{осн\ 2} = 40680$ грн.

Додаткова заробітна плата обчислюється як $ЗП_{дод} = 0,2 \cdot ЗП_{осн}$.

$ЗП_{дод\ 1} = 0,2 \cdot 32380 = 6476,00$ грн; $ЗП_{дод\ 2} = 0,2 \cdot 40680 = 8136$ грн.

Нарахування на фонд оплати праці (ФОП):

$ФОП_{ЄСВ} = 0,3677 \cdot ФЗП$

$ФОП_{ЄСВ1} = 0,3677 \cdot 32380 = 11906,126$грн;

$ФОП_{ЄСВ1} = 0,3677 \cdot 40680 = 13696,96$ грн.

Всього витрат:

$В_{ЗП1} = ЗП_1 + ФОП_{ЄСВ1} + ЗП_{дод1} = 50726,13$грн;

$В_{ЗП2} = ЗП_2 + ФОП_{ЄСВ2} + ЗП_{дод2} = 62512,96$грн.

З цієї суми утримуються обов’язкові відрахування на заробітну плату: єдиний соціальний внесок, який складає 3,6% від суми нарахованої заробітної плати та податок на доходи фізичних осіб, який складає 15% від суми нарахованої заробітної плати, зменшеної на суму єдиного внеску на загальнообов’язкове соціальне страхування та податкової соціальної пільги, військовий збір у розмірі 1,5%, від суми нарахувань. До окремих витрат також відносяться витрати на куповані вироби (матеріальне забезпечення) та спец обладнання для підтримки експерименту, накладні витрати. Витрати, що будуть супроводжувати проект розробки, порівнюватимемо в двох можливих підходах розробки.

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни (формула 4.1).

$$M_{Bi} = q_i \cdot p_i \ , \tag{6.1}$$

де q_i – кількість витраченого матеріалу i-го виду; p_i – ціна матеріалу i-го виду.

Матеріальні витрати в рамках проекту наведені в таблиці 6.2.2. Загальна сума матеріальних витрат становить 2449 гривень.

Таблиця 6.2.2 – Матеріальні витрати

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Флешка Kingston 4Gb	4	340,00	1360,00
Папір для друку А4, арк	500	0,158	79,00
Фарба для принтера	1	90,00	90,00
Дошка планування	1	750,00	750,00
Спиртовий маркер	10	17,00	170,00

Всього	2449,00

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою:

$$З_в = W * T * S, \quad (6.2)$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ =0.7 * 304 * 2.50 = 532 грн;

$З_{в2}$ =0.7 * 320 * 2.50 = 560 грн;

Розрахунок суми амортизаційних відрахувань. Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}} \quad (6.3)$$

де $C_Б$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год; $T_{ГОД}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 6.2.3 і рівна $C_Б$ = 36705 гривень. Річний робочий фонд часу приймемо за $T_{ГОД}$ = 2120 годин. З них реальний фактичний робочий час становить $T_{ФАК}$ = 304 години згідно об'єктно-орієнтованого підходу та $T_{ФАК}$ = 336 годин згідно процедурного підходу. Згідно вищезгаданої формули витрати на амортизацію становлять 1943,5 гривень та 2135,3 гривень для кожного підходу відповідно.

Таблиця 6.2.3 – Перелік необхідного обладнання

Найменування	Кількіст	Ціна,	Сума, грн

	ь, шт	грн	
Комп’ютер	2	11834,00	23668,00
Сервер HP Proliant DL 585 G6	2	5119,00	10238,00
Принтер А4 HP Laser	1	2799,00	2799,00
Середовища розробки	2	безкоштовно	безкоштовно
Операційна система	2	безкоштовно	безкоштовно
Всього більше 1000 грн.			36705,00
Всього витрат на амортизацію		3158,01	3490,44
Всього		39863,01	40195,44

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки наведені в таблиці 6.2.3.

Залежно від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників. Нехай вона буде дорівнювати 30%, що становить 11658,80 грн для об’єктно-орієнтованого і 14644,80 грн для процедурного підходу розробки.

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проекту, враховуючи всі вище описані затрати та нарахування. Цей вид

витрат складається з сум витрат на оплату праці (всього витрати на оплату праці), матеріальні затрати, затрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проекту.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.[38] Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво: 101118,06 грн використовуючи об'єктно-орієнтований підхід, 115273,50 грн при процедурному підході розробки.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{П}{Св} \tag{6.4}$$

де *П* – прибуток, *П= В - Св; Св* – собівартість.

У випадку даної розробки, маючи некомерційний проект без економічно корисного результату, можна прогнозувати, що економічна ефективність прямує до 0 у обох випадках. Однак це не є причиною для негативного економічного висновку щодо даного проекту, адже такого плану розробки приносять користь у вигляді інтелектуальних ресурсів, і, переважно, фінансуються або виконуються на замовлення організацій, зацікавлених в отриманні результатів досліджень та розробок[38]. Однак за надходження коштів на реалізацію ззовні можна вважати ефективність проекту рівною 1 (Е = 1), що означає перекриття витрат на розробку у повній мірі, тобто фінансування.

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($T_{ок}$):

$$T_{ок} = \frac{1}{E} \tag{6.5}$$

У нашому випадку прямого прибутку не існує. Прибуток можна прогнозувати на підприємствах, організаціях чи відомствах, що зацікавлені в дослідженні. Окупність же для розробки даного ПЗ за ефективності рівній одиниці можна вважати теж рівній 1 згідно формули 3.4.

Виходячи із експертних оцінок і складності програми, приймемо величину витрат на супровід і модернізацію програмного забезпечення, створеного за об'єктно-орієнтованим методом 25% (25279,52 грн) від початкових витрат, а за процедурним – 30% (34582,05 грн).

Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. Тобто у разі короткотермінової підтримки все ж за доцільніше обирати об'єктно-орієнтовану модель розробки, адже вартість короткотермінової підтримки згідно цієї моделі не вплине значно на сукупну вартість розробки.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Внаслідок змін система стає більш складною і погано керованою. Об'єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми, що значно скорочує подальші витрати на супровід і модернізацію [38].

Сумарні дані економічного розрахунку розробки даного проекту наведені в таблиці 6.2.4.

Таблиця 6.2.4 – Загальні витрати

Вид витрат	Об’єктно-орієнтований підхід, грн	Процедурний підхід, грн
Зарплата основна	34582,05	34582,05
Зарплата додаткова	6476,00	8136,00
Фонд заробітної плати	41058,05	42718,05
Відрахування на ФОП	11906,126	13696,96
Разом на оплату праці	60922,62	75078,06
Матеріальні витрати	2449,00	2449,00
Електроенергія	532,00	560,00
Амортизація	3158,01	3490,44
Накладні витрати	11658,80	14644,80
Обладнання	36705,00	36705,00
Разом на ін. витрати	39863,01	40195,44
Собівартість	101118,06	115245,50
Прибуток	відсутній	відсутній

Продовження таблиці 6.2.4 - Загальні витрати

Вартість розробленого ПЗ	**101118,06**	**115245,50**
Порівняльна економія витрат	23457,97	-
Загальні витрати (для споживача)	126397,58	149827,55
Порівняльна економія витрат (для споживача)	23457,97	-
Модернізація і супровід	25279,52	34582,05
Загальні витрати на розробку	**126397,58**	**149827,55**
Економія ($ЗВ_1$-$ЗВ_2$)		**23429,97**

Загальна вартість пропонованих робіт становить 149827,55 гривень для процедурного і 126397,58 гривень для об’єктно-орієнтованого підходів розробки. В даному випадку реалізації проекту варто вибрати об’єктно орієнтований підхід для розробки даного ПЗ, адже фінансово це більш вигідно. Також у даній методиці розробки кращі часові рамки та перспективи підтримки і модернізації. У оцінці вартості продукту варто враховувати можливість фінансування та спонсорування проекту, що дозволить гнучко та ефективно підійти до розробки та організації праці.

7. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Охорона праці

За такої умови, що деяка людина, котра проводить більшу частину робочого часу за розробкою розподіленого серверного програмного забезпечення для обрахунку великого об'єму інформації користувацького онлайн проекту на основі .NET Core, є працівником певного підприємства, їй слід виділити робоче комп'ютеризоване місце середньої важкості IIб за ДСН 3.3.6.042-99 з ВДТ [16].

Відповідно до вимог статті 5 Закону України „Про охорону праці”, та відповідно до п.2.1 наказу Міністерства соціальної політики України від 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», роботодавець повинен проінформувати працівників під розписку про умови праці та наявність на наших робочих місцях небезпечних та шкідливих виробничих факторів (фізичних, хімічних, біологічних, психофізіологічних), які виникають під час роботи з екранними пристроями та ще не усунуто, а також про можливі наслідки їх впливу на здоров’я працівників.

До обов'язків роботодавця, відповідно до п.2.2 наказу Міністерства соціальної політики України від 14.02.2018 № 207, також входить забезпечення навчання і перевірки знань працівників з питань охорони праці та безпечного використання екранних пристроїв до початку роботи з ними, а також у випадках модифікації та організації роботи обладнання.

Роботодавець за рахунок тривалості робочої зміни організує внутрішні регламентовані перерви для відпочинку відповідно до Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98, затверджених постановою Головного державного санітарного лікаря України від 10 грудня 1998 року № 7 (далі – ДСанПІН 3.3.2.007-98) [18].

Роботодавець забезпечує за свій рахунок проведення медичних оглядів працівників відповідно до вимог Порядку проведення медичних оглядів працівників певних категорій, затвердженого наказом Міністерства охорони здоров'я України від 21 травня 2007 року № 246, зареєстрованого в Міністерстві юстиції України 23 липня 2007 року за № 846/14113 [19].

Відповідно до п.2.13 ДСН 3.3.6.042-99, роботодавець проводить попередні (при прийомі на роботу) та періодичні медичні огляди в процесі роботи відповідно з діючим наказом МОЗ України.

Приміщення для роботи з ЕОМ розташоване не у підвальних приміщеннях та не на цокольних поверхах. Тобто, робоче місце відповідає п.2.2 ДСанПіН 3.3.2-007-98.

Відповідно до п.2.3 ДСанПіН 3.3.2-007-98, площа на одне робоче місце становить 8 $м^2$, а об’єм – 25 $м^3$.

Природне освітлення здійснюється через світлові прорізи, орієнтовані переважно на північ та північний схід і забезпечують коефіцієнт природної освітленості (КПО) 3%. Тобто, КПО робочого місця відповідає п.2.5 ДСанПіН 3.3.2-007-98.

Відповідно до п.2.9 ДСанПіН 3.3.2-007-98, віконні прорізи приміщень для роботи з ВДТ обладнані регульованими пристроями (жалюзі, завіски, зовнішні козирки).

Відповідно до п.2.15 ДСанПіН 3.3.2-007-98, у робочому приміщенні щоденно робиться вологе прибирання.

Відповідно до п.2.16 ДСанПіН 3.3.2-007-98, робоче приміщення оснащене аптечками першої медичної допомоги.

Зазначення освітлення освітленості на поверхні робочого столу в зоні розміщення документів становить 400 лк. Тобто, освітленість робочого місця відповідає п.3.2.3 ДСанПіН 3.3.2-007-98.

Конструкція робочого місця забезпечує підтримання оптимальної робочої пози, та відповідає п.4.2 ДСанПіН 3.3.2-007-98.

Робоче місця працівника з екранними пристроями облаштовується таким устаткуванням, яке не створює зайвого шуму та не виділяє надлишкового тепла. Рівні шуму на робочих місцях осіб, які працюють з екранними пристроями, відповідають вимогам Санітарних норм виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99, затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року № 37.

В приміщенні з робочими місцями середня температура повітря +20°С, вологість повітря 50%, швидкість повітря 0,2 м/с, тобто вона відповідає п.1.1.1 ДСН 3.3.6.042-99.

Згідно з п.1.2.2 ДСН 3.3.6.042-99, температура внутрішніх поверхонь робочої зони (стіни, підлога, стеля) не виходить більш ніж на 2°С за межі оптимальних величин температури повітря для даної категорії робіт, тобто приміщення відповідає п.1.1.2 ДСН 3.3.6.042-99. В холодний період року в приміщеннях з робочими місцями температура повітря +20°С, відносна вологість 75%, швидкість руху повітря – 0,2 м/с. В теплий період року в приміщеннях з робочими місцями температура повітря +25°С, відносна вологість 70%, швидкість руху повітря – 0,3 м/с. Таким чином робоче місце відповідає п.2.3 ДСН 3.3.6.042-99.

Відповідно до п.2.4 ДСН 3.3.6.042-99, приміщення з робочими місцями з надлишком (явного) тепла використовує природну вентиляцію (аерацію).

Вимірювання параметрів мікроклімату на робочих місцях по розробці розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core проводяться на висоті 0,5 – 1,0 м від підлоги – при роботі сидячи, 1,5 м від підлоги – при роботі стоячи, таким чином відповідаючи п.3.3 ДСН 3.3.6.042-99.

Відповідно до п.3.6 ДСН 3.3.6.042-99, температура та відносна вологість повітря вимірюються приладами, заснованими на психрометричних

принципах. Можливе використання тижневих і добових термографів і гігрографів.

Відповідно до п.3.7 ДСН 3.3.6.042-99, швидкість руху повітря вимірюється анемометрами ротаційної дії. Малі величини швидкості руху повітря (менше 0,3 м/сек.), особливо при наявності різноспрямованих потоків, вимірюються електроанемометрами, циліндричними або кульовими кататермометрами.

Відповідно до п.3.8 ДСН 3.3.6.042-99, температура поверхонь огороджуючих конструкцій (стін, стелі, підлоги) або обладнань (екранів і т. ін.) вимірюються приладами, що діють за принципом термоелектричного ефекту.

В заключення можна сказати, що розглянуті стандарти охорони праці на робочих місцях розробки розподіленого серверного програмного забезпечення для обрахунку великого об'єму інформації користувацьких онлайн проектів на основі .NET Core відповідає усім розглянутим основним нормативам для робочих місць даного типу.

7.2 Безпека в надзвичайних ситуаціях

Під час розробки розподіленого серверного програмного забезпечення для обрахунку великого об'єму інформації користувацьких онлайн проектів на основі .NET Core розробник зіштовхується з багатьма факторами, котрі впливають на його продуктивність під час виконання завдання.

Трудова діяльність користувачів комп'ютерів (ВДТ) відбувається у певному виробничому середовищі, яке впливає на їх функціональний стан. Найбільш значимі — фізичні фактори виробничого середовища, до яких належать електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ціла низка світлотехнічних показників. Вплив хімічних та, особливо, біологічних факторів виробничого середовища на користувачів комп'ютерів — значно менший.

Трудовий процес суттєво впливає на психофізіологічні можливості користувачів комп'ютерів, оскільки їх діяльність характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп'ютерів відзначається значними інформаційними навантаженнями.

Професійні якості та виробничий досвід, які визначають внутрішні засоби діяльності, обумовлюють надійну та безпомилкову діяльність користувачів комп'ютерів, дозволяють знаходити безпечні методи розв'язання виробничих завдань навіть у нестандартних ситуаціях.

7.2.1 Особливості роботи користувачів комп'ютерів

У професійних операторів частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, серцево-судинної, імунної та статевої систем, захворювання шкіри. Зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів.

Сучасна професія користувача ВДТ належить до розумової праці, яка характеризується: високою напруженістю зорових функцій; одноманітною позою; великою кількістю стереотипних висококоординованих рухів, що виконуються лише м'язами кистей рук на фоні малої загальної рухової активності; значним нервово-емоційним компонентом, особливо в умовах дефіциту часу; роботою з великими масивами інформації, що викликає активізацію уваги та інших вищихпсихічних функцій. Крім того, при роботі з дисплеями на електронно-променевих трубках виникає вплив на користувача цілої низки факторів фізичної природи — електростатичні поля, радіочастотне та рентгенівське випромінювання тощо.

Діяльність професіоналів можна поділити на три групи:

1. Діяльність, яка пов'язана з виконанням нескладних багаторазово повторюваних операцій, що не вимагають великого розумового напруження. Наприклад, робота операторів комп'ютерного набору, працівників довідкових служб.
2. Діяльність, яка пов'язана із здійсненням логічних операцій, що постійноповторюються. Це робота інженера-економіста, інженера-проектувальника, оператора автоматизованого виробництва.
3. Діяльність, коли в процесі роботи необхідно приймати рішення за відсутності заздалегідь відомого алгоритму. Наприклад, робота інженера-програміста, диспетчерів руху залізничного транспорту, аеропортів тощо.

У користувачів, які інтенсивно використовують комп'ютер в умовах значних розумових напружень досить часто (40—70%) виникають психологічні та поведінкові порушення (нервозність, роздратування, тривога, нерішучість, замкнутість тощо).

Інформаційне перевантаження користувачів ВДТ супроводжується низкою специфічних захворювань, які називають інформаційними. Першим симптомом їх є головний біль. Дослідження, проведені в США, Німеччині, Швейцарії та інших країнах, показали, що робота з обслуговування ВДТ супроводжується підвищеним напруженням зору, інтенсивністю і монотонністю праці, збільшенням статичних навантажень, нервово-психічним напруженням, впливом різного виду випромінювань та ін.

Вважається, що стан організму операторів ВДТ визначається комплексним впливом факторів трудового процесу і середовища, значення яких є неоднаковим. На операторів з малим стажем роботи на ВДТ домінуючий вплив чинять фактори середовища, а на операторів зі стажем понад 5 років - фактори трудового процесу.

7.2.2 Розлади здоров'я користувачів, що формуються під впливом роботи за комп'ютером

Комп’ютерний зоровий синдром (КЗС) – комплекс порушень здоров’я, який може виникати у користувачів персональних комп’ютерів (ПК). Діагноз ставлять, якщо людина, що працює за ПК протягом двох годин, висловлює хоча б дві з десяти скарг:

головний біль

- сльозотеча
- різь
- туман
- двоїння
- свербіж
- важкість в очах
- фотофобія
- миготіння знаків на екрані
- нудота.

У користувачів ПК дуже поширені кон’юнктивіти і блефарити, патогенетично пов’язані з КЗС.

Синдром розвивається при умові, що робоче місце організовано неправильно – у користувача незручне крісло, відсутні пюпітри для паперів, підставки для ніг та кистей рук, не встановлена висота і нахил монітора відносно очей, відстань від очей до екрана. За таких умов тіло людини при роботі займає вимушене положення: спина статично напружена, шия витягнута, плечі жорстко фіксовані. Напружені м’язи погіршують кровотік у сонних артеріях, а недостатнє кровозабезпечення головного мозку веде до очманіння, появи головного болю. На фоні шийного остеохондрозу з’являється відчуття випирання очних яблук, туману в очах, мушок та райдужних кіл у полі зору. Розвитку КЗС сприяє поганий мікроклімат приміщення, значна загальна іонізація та мікробне забруднення, а також куріння.

Діяльність користувачів комп'ютерів характеризується тривалою багатогодинною (8 год. і більше) працею в одноманітному напруженому сидячому положенні, малою руховою активністю при значних локальних динамічних навантаженнях, що припадають лише на кисті рук. Такий характер роботи може призвести до появи низки хворобливих симптомів, що об'єднані загальною назвою — синдром довготривалих статичних навантажень (СДСН). Узагальнюючи статистичні дані можна зробити висновок про те, що СДСН може проявлятись втомою, скутістю, болем, судомою, онімінням та ін., локалізуватись у різних частинах тіла (шия, спина, руки, ноги та ін.) і виникати індивідуально з різною частотою (ніколи, рідко, епізодично, щоденно).

Оператори по введенню даних частіше скаржились на біль у руках, шиї та у верхній частині ніг, тоді як оператори діалогового режиму — на біль спини (частіше у поперековому відділі хребта) та плечового суглоба.

Тривала робота за комп'ютером при неправильному, з фізіологічної точки зору, положенні тіла може викликати такі вади постави, як сутулість, викривлення хребта (сколіоз) та ін.

До найбільш частих симптомів, що характерні захворювань кистей рук належить:

- больові відчуття різної сили у суглобах та м'язах кистей рук;
- оніміння та повільна рухливість пальців;
- судоми м'язів кисті;
- поява ниючого болю в ділянці зап'ястка.

Несприятливий умови роботи за комп'ютером впливають на серцево-судинну систему. Пов'язують це з гіподинамією. В умовах обмеження м'язової активності, коли зменшується потреба тканин у кисні та субстрактів біологічного окислення, можна було б очікувати зниження напруженості функції серцево-судинної системи. Однак цього не відбувається; навпаки, розвивається детренованість серцево-судинної системи, зростає частота

серцевих скорочень в стані спокою. Навіть при незначних, короткочасних фізичних навантаженнях пульс досягає 100 і більше ударів за хвилину.

Тривале обмеження навантаження на м'язовий апарат може стати причиною функціональних порушень, а в деяких випадках призвести до виникнення атеросклерозу, аритмії, гіпертонічної хвороби, інфаркту міокарда.

Вказується на збільшення відсотку хвороб органів травлення у осіб, які інтенсивніше використовували ВДТ. Частіше за інші форми відзначені хронічні гастрити та холецистити. Висловлено припущення, що у формуванні таких захворювань визначальна роль належить нервово-емоційним напруженням.

На завершення, слід добавити що окрім факторів на робочому місці, котрі впливають на функціональний стан користувача комп'ютера, є також фактори, котрі впливають поза робочим місце (стан екології, повсякденний стрес, соцільні проблеми та ін.)

ВИСНОВКИ

Отже, дана науково-інженерна робота, що була виконана під керівництвом к.ф.-м.н., доцента Бойка Ігоря Володимировича в Тернопільському національному технічному університеті імені Івана Пулюя успішно завершена. В цьому звіті було викладено процес і результат діяльності програмного інженера, з темою магістерської роботи "Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core".

Під час виконання магістерської роботи магістра переді мною стояла задача реалізувати розподілене серверне програмне забезпечення користувацького проекту(сайту) для компанії і таким чином вийти на інтернет-ринок, тим самим спростивши процес реалізації товару. Для початку, я детально проаналізував предметну область, з якою повинен був працювати, вивчивши критерії формування характеристик товару. Після цього, етап реалізації даного сайту був розділений на менші та логічні блоки. В результаті, процес роботи був розподілений на етапи та логічні блоки. Варто згадати, що в якості магістерської роботи переді мною стояла задача, щоб реалізувати функціонал даного сайту, використовуючи фреймворк .NET Core, тобто описати сценарій дій для кожного з варіанту використання, що були виявлені мною для даного приватного веб-магазину на етапі аналізу предметної області.

Отже, на момент завершення магістерської роботи магістра мною було виконано близько 100% від всіх виявлених раніше функцій даного розподіленого серверного ПЗ сайту. Таким чином я спроектував і розробив розподілене серверне ПЗ для обробки великого об'єму інформації на основі .NET Core з магістерської роботи. Цього було цілком достатньо щоб наділити сайт базовими функціями.

В процесі виконання роботи було в повній мірі виконано поставлене завдання, щодо серверного програмного забезпечення на основі фреймворку

.NET Core. В результаті виконання даного завдання було отримано практичні навички щодо створення сайту для користувацького проекту. Також, виконання даного завдання дозволило засвоїти всі набуті протягом навчання навички на прикладі реального проекту.

Щодо оцінки отриманих результатів, то можна відмітити, що всі вище перечисленні навички в сучасному світі є дуже корисними, оскільки все на сьогоднішній день відбувається у веб-мережі. Саме тому вміння створювати подібні серверні застосунки є дуже корисним для більшості інженерів програмного забезпечення.

Результат виконання поставленого завдання під час проектування та розробки серверного ПЗ відповідає окремій галузі в сфері інформаційних технологій, а саме створення розподілених серверних систем сайтів(таких як Amazom, Aliexpress), що на сьогоднішній день набуває все більшої популярності. Вміння створення великомасштабних сайтів на даний момент стає все більш затребуваним в даній сфері, а все тому, що на даний момент в усіх сферах відбувається процес переходу до інтернет-вендорингу. Іншими словами, відбувається масовий перехід усього в мережу інтернет, за допомогою цих самих сайтів. Отже, на сьогоднішній день, отримані навички під час виконання магістерської роботи магістра користуються великим попитом на IT ринку та направлені на всі галузі.

Отже, згідно всього вище написаного можна зробити наступний висновок щодо значимості даної роботи яка була виконана мною протягом інженерного проектування та реалізації розподіленого серверного програмного забезпечення для веб-ферми. Сайт для будь-якого магазину має дуже велике значення, оскільки саме він надає уявлення про компанію, наявний в неї товар та послуги. Крім того, веб-магазин покращує досвід клієнта, що призведе до формування гарного враження від магазину, що в свою чергу приведе цього клієнта знову до веб-магазину. Саме тому вміння коректно та відповідно до усіх вимог створювати сайти для будь-якої потреби на сьогоднішній дуже ціниться роботодавцями в плані IT-сфери.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The vital guide to modern programming languages and their uses [Електронний ресурс]. – 2008. – Режим доступу до ресурсу: https://searchapparchitecture.techtarget.com/definition/object-oriented-programming-OOP.
2. Офіційна документація по C# [Електронний ресурс] – Режим доступу.: https://msdn.microsoft.com/uk-ua/library/618ayhy6.aspx
3. Офіційна документація по технології .NET Framework [Електронний ресурс] – Режим доступу.: https://docs.microsoft.com/uk-ua/dotnet/
4. Г.Ю. Громов. Введення в реляційні бази даних – 2009. – 254 с.
5. І.О. Завадський. Основи баз даних – 2011. – 192 с.
6. .NET Core Guide [Електронний ресурс] // 2019 – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/core/.
7. Parallel Computer Architecture - Models [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/ parallel_computer_architecture/ parallel_computer_architecture_models.htm.
8. Introduction to Parallel Computing [Електронний ресурс] – Режим доступу до ресурсу: https://computing.llnl.gov/tutorials/parallel_comp/.
9. File storage systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/support/knowledgecenter/en/SSQRB8/com.ibm.spectrum .si.doc/tpch _r_storagesystem_file.html.
10. Nuncic M. The Evolution of Storage: File Storage vs. Block Storage vs. Object Storage – Part 1 [Електронний ресурс] / Michael Nuncic. – 2018. – Режим доступу до ресурсу: https://www.ontrack.com/blog/2018/02/22/the-evolution-of-storage-file-storage-vs-blockstorage-vs-object-storage-part-1/.
11. Introduction to Storage Area Networks / J.Tate, P. Beck, H. Ibarra, L. Miklas., 2017. – 300 с. – (1-ше). – (9780738442884).

12. What is an API? (Application Programming Interface) [Електронний ресурс] – Режим доступу до ресурсу: https://www.mulesoft.com/resources/api/what-is-an-api.
13. Gartner: Top 10 cloud storage providers [Електронний ресурс] // Network World. – 2013. – Режим доступу до ресурсу: https://www.networkworld.com/article/2162466/cloudcomputing/cloud-computing-gartner-top-10-cloud-storage-providers.html.
14. Gan C. How to build a Network Attached Storage (NAS) / Chin Gan., 2016. – 84 c. – (B01BU2NTO0).
15. Pessach D. Distributed Storage: Concepts, Algorithms, and Implementations / Distributed Storage: Concepts, Algorithms, and Implementations Pessach., 2013. – 106 c. – (1-ше). – (978-1482561043).
16. Shrivastava A. Information Storage and Management: Storing, Managing, and Protecting Digital Information / Alok Shrivastava., 2009. – 106 c. – (1-ше). – (9780470294215).
17. Swan M. Blockchain: Blueprint for a New Economy / Melanie Swan., 2015. – 152 c. – (1).
18. Karlsson K. C#—UnitOfWork And Repository Pattern [Електронний ресурс] / Kristoffer Karlsson. – 2017. – Режим доступу до ресурсу: https://medium.com/@utterbbq/cunitofwork-and-repository-pattern-305cd8ecfa7a.
19. Martinez J. Understanding Proof-of-Work [Електронний ресурс] / Julian Martinez. – 2018. – Режим доступу до ресурсу: https://medium.com/@julianrmartinez43/understandingproof-of-work-part-1-586d7ee6b014.
20. Morris K. How Much Does A 51% Attack Cost? [Електронний ресурс] / Kai Morris. – 2018. – Режим доступу до ресурсу: https://cryptodisrupt.com/how-much-does-a-51- attack-cost/. 109

21. C# Coding Conventions [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-aprogram/coding-conventions.
22. Hanmer R. Pattern-Oriented Software Architecture For Dummies / Robert Hanmer. – USA: For Dummies, 2013. – 384 с. – (1).
23. Storj: A Decentralized Cloud Storage Network Framework [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://storj.io/storj.pdf.
24. Vorick D. Sia: Simple Decentralized Storage [Електронний ресурс] / D. Vorick, L. Champine. – 2014. – Режим доступу до ресурсу: https://sia.tech/sia.pdf.
25. Drake N. Best cloud storage of 2018 : Free, paid and business options [Електронний ресурс] / Nate Drake. – 2018. – Режим доступу до ресурсу: https://www.techradar.com/news/the-best-cloud-storage.
26. HTTP Over TLS [Електронний ресурс]. – 2000. – Режим доступу до ресурсу: https://www.rfc-editor.org/info/rfc2818.
27. Introduction to Parallel Computing [Електронний ресурс] // 2019 – Режим доступу до ресурсу: https://computing.llnl.gov/tutorials/parallel_comp/.
28. Форма №2 “Звіт про фінансові результати”: методика підготовки[Електронний ресурс]. – Режим доступу: URL: http://osvita.ua/vnz/reports/accountant/17368/.
29. Parallel Computer Architecture - Models [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.tutorialspoint.com/parallel_computer_architecture/parallel_computer_architecture_models.htm.
30. Parallel Programming in .NET [Електронний ресурс] // 2019 – Режим доступу до ресурсу: https://docs.microsoft.com/en-gb/dotnet/standard/parallel-programming/.
31. Videla A. RabbitMQ in Action: Distributed Messaging for Everyone / A. Videla, J. Williams., 2012. – 312 с. – (1-ше). – (978-1935182979).
32. SCALABILITY: SCALE-UP OR SCALE-OUT, WHAT IT IS AND WHY YOU SHOULD CARE [Електронний ресурс]. – 2013. – Режим доступу до

ресурсу: https://www.brianjgraf.com/2013/05/17/scalability-scale-up-scale-out-care/.

33. Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/s3/.
34. Microsoft Azure [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/.
35. Google Cloud Platform [Електронний ресурс] – Режим доступу до ресурсу: cloud.google.com.
36. Smith J. Entity Framework Core in Action / Jon Smith., 2018. – 520 с. – (1-ше). – (978- 1617294563).
37. Bai H. Programming Microsoft Azure Service Fabric / Haishi Bai., 2018. – 528 с. – (2-ге). – (978-1-5093-0709-8).
38. Методичні вказівки для виконання розділу дипломної роботи щодо техніко-економічного обґрунтування вибору проектного рішення розробки та оцінки якості програмного забезпечення / Упор. Петрик М.Р., Кінах Я.І., Головатий А.І., Рогатинська Л.Р. – Тернопіль: Вид-во ТНТУ ім. І. Пулюя. – 2013. – 34 с.
39. Host ASP.NET Core in a web farm [Електронний ресурс] // 2019 – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/web-farm?view=aspnetcore-3.1.

ДОДАТКИ

Додаток А - Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра “Програмної інженерії”

ЗАТВЕРДЖУЮ

_____

“___ “_____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломної роботи магістра

“Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core ”

ПІБД 16. № СПм-18-184. 001 ТЗ

Розробники

Керівник проекту

_____

“ _____” 2019р.

Виконавець

студ. Юрченко К.Р.

“ _____” 2019р.

Тернопіль 2019

ЗМІСТ

Вступ

1. Підстави до розробки
2. Призначення до розробки
3. Вимоги до програмного продукту
   3.1 Функціональні характеристики
   3.2 Склад та параметри технічних засобів
   3.3 Інформаційна та програмна сполучність
4. Стадії розробки
5. Програмна документація
6. Порядок контролю та приймання

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до календарного плану виконання дипломної роботи магістра на 2019р. за спеціальністю 121-Інженерія програмного забезпечення.

Тема роботи: “Розробка розподіленого серверного програмного забезпечення для обробки великого об'єму інформації користувацьких онлайн проектів на основі .NET Core ”.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний продукт призначений для надання потрібної інформації в сфері веб-маркетингу та продажів продукції. Дане програмне забезпечення буде розроблене для серверів, котрі будуть розміщуватись в одному кластері, а отже ПЗ буде розподіленим з можливістю розміщення на кластері (веб-фермі) та одночасним доступ до спільної бази даних.

Мета розробки полягає в створені програмного забезпечення, що дозволить покупцям отримувати перелік категорій, товарів, додавати товари до кошику, оформлювати та оплачувати замовлення. Адміністратор має можливість змінювати відображувані категорії, товари, змінювати статус та завершення замовлення.

Даний продукт дозволить користувачам значно економити час при пошуку необхідної інформації, а також під час її зберігання, що дозволить потенційним клієнтам знаходити необхідні їм товари.

3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні характеристики

Програмний продукт має забезпечити виконання наступних дій:

- Дозволяє клієнтові переглянути список наявних товарів на сайті;
- Клієнт має змогу переглянути список категорій, на які поділені всі товари, що реалізуються на даному сайті;
- Надає можливість клієнтові переглянути інформацію про окремий товар, його фотографію;
- Дозволяє додати обраний товар до кошика, щоб в подальшому оформити покупку даного товару;
- Надає можливість клієнтові переглянути список товарів, що знаходяться в кошику;
- Дозволяє клієнтові оформити замовлення, та завершити купівлю обраних ним товарів;
- Клієнт може вибрати метод оплати та оплатити товар з допомогою веб-банкінгу;
- Надає можливість клієнтові зареєструвати аккаунт на сайті;
- Клієнт має можливість змінити чи додати нову інформацію в особистому кабінеті;
- Клієнт має можливість авторизації з допомогою особистого кабінету;
- Адміністратор має змогу переглянути весь список замовлень, які були оформлені раніше;
- Надає можливість адміністратору сайту обробляти окреме замовлення, переглядаючи інформацію про дане замовлення;

- Дозволяє адміністратору змінити статус замовлення, безпосередньо його опрацювання;
- Адміністратор може створювати опитування, додавати нові рекламні слайдери на сторінки сайту;
- Адміністратор може закрити магазин для покупок.

3.2 Склад та параметри технічних засобів

Функціонування програми забезпечується: інфомат з розрядністю процесора x64, з 4 Гб оперативної пам‘яті, встановленою системою Windows 7/10/Server 2008+, та не менш 400 Мб вільного місця на жорсткому диску.

3.3 Інформаційна та програмна сполучність

Програмний продукт повинен коректно функціонувати під керуванням ОС Microsoft Windows 7/10/Server 2008+. Програмний продукт має назву “Веб-сайт «FiberDock»”. і повинен бути написаний мовою C# з використанням фреймворку .NET Core.

4. СТАДІЇ РОЗРОБКИ

- - аналіз предметної галузі;
- - вибір та проектування бази даних;
- - розробка прикладного програмного забезпечення;
- - тестування програмного продукту;
- - оформлення супровідної документації;
- - здача продукту.

5. ПРОГРАМНА ДОКУМЕНТАЦІЯ

Для програмного продукту повинні бути розроблені наступні документи:

- Специфікація;
- Технічне завдання;
- Пояснювальна записка;

6. ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Розроблений програмний продукт має виконувати всі вимоги, що складаються з перерахованих у п.3.1 характеристик.

Приймання проводиться спеціально створеною комісією в термін до 24 червня 2019р.

УДК 004.75

К.Р. Юрченко – магістрант
Тернопільський національний технічний університет імені Івана Пулюя, Україна
І.В. Бойко – кандидат фізико – математичних наук, доцент
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА РОЗПОДІЛЕНОГО СЕРВЕРНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КОРИСТУВАЦЬКИХ ОНЛАЙН ПРОЕКТІВ

K.R. Yurchenko – magistrate, I.V. Boyko – Ph.D, Assoc. Prof.

DEVELOPMENT OF DISTRIBUTED SERVER SOFTWARE FOR CUSTOM ONLINE PROJECTS

Більшість веб-проектів починає своє життя з невеликої кімнати і маленького сервера. Очевидно, розробники сподіваються на успіх свого дітища. Але коли приходить популярність, то разом з нею з'являються і додаткові проблеми. Найчастіше вони виражаються в тому, що сервер перестає справлятися з навантаженням. Існує кілька способів вирішення цього питання - оптимізація графіки і скриптів, нарощування потужності сервера, створення розподіленого сервера, що складається з декількох комп'ютерів. Кожен з методів має свої першочергові переваги, але перші два методи мають межу вдосконалення потужності, котру можна подолати лише третім способом.

Третій спосіб підвищення продуктивності веб-сервера - використання "кластера" з двох або більше комп'ютерів[1]. Це рішення менш популярно, ніж використання багатопроцесорних технологій. І це дивно, оскільки "кластер" має ряд переваг перед одним потужним сервером.

На перший погляд здається, що просто докупити і встановити на сервер додатковий процесор набагато дешевше, ніж придбати нову машину. Але так думають тільки ті, що ніколи не стикався з цією проблемою на практиці. Насправді мало просто встановити новий процесор в слот. Для підтримки багатопроцесорної структури потрібно багато додаткових пристроїв і додатків. І це ще не все. Збільшення кількості процесорів часто вимагає нарощування обсягів оперативної пам'яті.

У той же час установка нового сервера може обійтися не так дорого. Адже ніхто не примушує вас купувати потужну машину. Для розвантаження основного сервера цілком можна використовувати звичайний ПК, який просто візьме на себе меншу частину запитів. А якщо в майбутньому такий "кластер" перестане справлятися, то можна поступово покращувати другий сервер, збільшуючи паралельно його завантаження. Таким чином, використання "кластера" дозволяє витрачати кошти не відразу, а поступово.

Ще одним плюсом розподіленого веб-сервера перед багатопроцесорним є продуктивність. Почати потрібно з того, що більшість сучасних серверів має обмеження в 16 процесорів. І це ще хороший варіант. Багато ж задовольняються серверами, в які не можна встановити більше двух процесорів. У той же час в "кластері" можуть брати участь до 255 машин, кожна з яких також може бути багатопроцесорною. Таким чином, потужність всієї системи зростає практично безмежно. Крім того, не можна забувати, що в багатопроцесорному сервері всі системні ресурси знаходяться в загальному користуванні. Таким чином, обмежувачем його продуктивності може стати, наприклад, оперативна пам'ять. "Кластер" позбавлений цієї проблеми. Кожен комп'ютер в складі розподіленого сервера має власні системні ресурси, обсяги яких можуть визначати навантаження на машину.

Важливою особливістю розподіленого сервера є гнучкість. Адже в нього можуть входити машини з різним апаратним забезпеченням, операційними системами та програмним забезпеченням. На перший погляд, здається, що це не потрібно[2]. Насправді,

ця особливість дуже допомагає при виборі операційної системи і ПО для сервера. Досить встановити на різні машини різні варіанти і простежити за їх працездатністю. Крім того, гнучкість дозволяє не прив'язуватися до комп'ютерного обладнання певної фірми і в майбутньому здійснювати дешевші апгрейди. Адже ті пристрої, які сьогодні є найсучаснішими і дорогими, через рік будуть набагато дешевше. Ну, а якщо ціна "заліза" не грає великої ролі, то власники серверу можуть часто оновлювати свої машини, маючи, таким чином, постійно покращуваний розподілений сервер.

Крім того, використання одного сервера залишає відкритим питання проведення профілактичних робіт. Адже в цьому випадку комп'ютер доведеться вимикати. Таким чином, сайт під час проведення робіт залишається недоступним. Зменшити потенційні втрати від перерв можна тільки одним способом: відключати сервер ночами, в період мінімальної відвідуваності. Однак це навряд чи сподобається технічного персоналу, який обслуговує комп'ютери. А у випадку з розподіленим сервером такої проблеми просто не існує. Машини можна зупиняти по черзі, так що відвідувачі сайту в найгіршому випадку помітять лише деяке уповільнення роботи.

Але, звичайно ж, не буває нічого ідеального. У розподіленого сервера, в порівнянні з багатопроцесорним, є один істотний мінус. Він полягає в тому, що за кожен комп'ютер, встановлений у хостинг-провайдера, доведеться платити додаткові гроші. Звичайно ж, це дуже серйозний недолік. Проте він цілком покривається плюсами розподілених серверів, а тому використання цього варіанту для серйозних проектів з великою аудиторією є більш ніж виправдано.

Література

1. Клеппман М. Високонавантажені додатки. Програмування, масштабування, підтримка / М. Клеппман. – Санкт-Петербург: "Питер", 2019. – 640 с.
2. Бьорнс Б. Розподілені системи. патерни проектування / Брендан Бьорнс. – Санкт-Петербург: "Питер", 2019. – 224 с.