

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

**Побудова багаторівневого веб-застосування на
платформі Docker-контейнерів**

**Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки та
інформаційні технології» - 122**

Керівник курсової роботи

к.т.н., ст. викладач

Черкасов Д. І.

(підпис)

“ ____ ” _____ 2020 р.

Виконав студент 3 курсу

Іщенко І.О.

“ ____ ” _____ 2020 р.

Зміст

1. ВСТУП.....	3
2. Огляд Існуючих Рішень.....	5
2.1. Огляд архітектури застосунку.....	6
2.1.1. Однорівнева архітектура.....	6
2.1.2. Дворівневі застосування.....	7
2.1.3. Три-рівневі застосування.....	8
2.1.4. Багаторівневі застосування	9
2.1.5. Порівняльний аналіз	9
2.2 Огляд варіантів розміщення	11
2.2.1. Оренда дата-центру з розміщенням фізичного обладнання	11
2.2.2. Розміщення на власних ресурсах	12
2.2.3. Розміщення на хмарних ресурсах.....	13
2.2.4. Порівняльний аналіз способів розміщення	15
3. Розробка власного рішення	17
3.1 Структурна схема архітектури застосунку	18
3.2 Опис компонентів застосунку	20
3.2.1. Компонент керування продуктами(CouseWork_Online_Shop_Backend_Product).....	20
3.2.2. Компонент керування категоріями (CourseWork_Onlile_Shop_Backend_Category)	21
3.2.3 Компонент зв'язку з базою даних(DbManager)	22

3.2.4	Компонент користувачького інтерфейсу (CourseWork_Online_Shop_Frontend)	23
3.2.4.1.	Авторизація та аутентифікація	24
3.2.4.2.	Сервіс керування користувачами	24
3.2.4.3.	Сервіс створення покупки	25
3.4	Розробка компонента користувачького інтерфейсу (CourseWork_Online_Shop_Frontend)	28
3.5	Загальна розробка веб-застосунку	29
4.	Висновки	36
5.	Список використаної літератури	37

1. ВСТУП

На сьогоднішній день, актуальність on-line шопінгу зростає щодня. Особливо зараз, коли увесь світ охоплений новою пандемією та ми знаходимось на карантині, виникає потреба в покупці необхідних товарів саме сидячи вдома. У виниклій ситуації, люди все частіше обирають саме on-line шопінг. Для того, щоб така платформа користувалась попитом, вона має підлягати під такі критерії :

- Застосунок має бути цілодобово працездатним та витримувати навантаження від великої кількості користувачів;
- Можливість оновлювати програмний продукт без втрат;
- Застосунок повинен мати спроможність зберігати дані про своїх користувачів в захищеному та зашифрованому вигляді;
- Підтримувати зв'язок з клієнтами через засоби комунікації.

Для реалізації вищесказаних вимог, можна звернутись до концепції мікросервісної архітектури, яка полягає в розподілених обчисленнях, що дозволяє збільшити продуктивність та швидкість роботи на кожному рівні. Згадану концепцію можна втілити за допомогою контейнеризації. Існує

багато платформ для розробки застосунків на основі контейнерів. Однією з найпопулярніших платформ є Docker.

На сьогодні, розробка веб-застосунків займає одну з лідуючих позицій серед програмних продуктів через великий попит та швидкість у розробці. Тому розробка on-line магазину на основі платформи Docker контейнерів стала темою моєї курсової роботи.

Основною метою моєї курсової роботи є показати основні переваги розробки модульних веб-застосунків на основі контейнеризації. Адже гнучкість, швидкість та потужність є невід'ємними ознаками програмного продукту.

Актуальність роботи полягає в тому, що сьогодні все більше застосунків переходять на модульну структуру. Такий підхід має багато переваг і зручний у розробці в команді будь-якого розміру. Можливість використати ці практичні навички підіймає фахівця на новий рівень.

Суть моєї роботи полягає в наступному :

- Розробити веб-застосунок on-line магазину, який зможе витримувати великі навантаження, та одночасний онлайн до 1 тис. одночасно активних користувачів та до 1 тис. торговельних транзакцій за секунду.
- Застосунок має бути 365/24/7 (оновлення функціоналу не мають переривати працездатність застосунку).

- Інформація про користувачів має бути в захищеному вигляді
- Користувачі мають бути в курсі оновлень через засоби комунікації. Наприклад : електронна пошта.

2. Огляд Існуючих Рішень

Реалізація онлайн магазину на платформі контейнерів значно полегшує розробку компонентної структури застосування. Адже використання контейнерів дозволяє спростити процес розгортки, розробки, поширення та роботи веб-застосунку. Це досягається за допомогою пакування кожної компоненти структури застосунку в окремий контейнер, який має ізольоване та захищене середовище. Така середа дозволяє спростити етап розгортки та тестування роботи кожної з компонент, не впливаючи на інші частини застосунку. Зазвичай такі компоненти поділяють на 3 рівні :

- Front-end – або клієнтська частина застосунку, де ми отримуємо кінцевий результат представлення даних, отриманих від сервера.
- Middleware – або бізнес-логіка, де обробляються дані застосунку, які отримані від сервера.
- Back-end – або серверна частина, яка обробляє запити до бази даних та передає дані в чистому вигляді до бізнес-

логіки, або напряду до клієнтської частини (у випадку з однорівневою архітектурою)

Використання контейнерів додає застосунку гнучкості, через відмовостійкість, та можливість взаємо-заміни кожного з контейнерів. Використання платформи Docker контейнерів надає змогу легко керувати існуючими контейнерами, та перевіряти їх роботу. Використання контейнерів дозволяє розгорнути застосунок з використанням ядра ОС Linux задля розподілення задачі на велику кількість розподілених компонент, які виконують свою частину роботи.

В цьому розділі я пропоную оглянути існуючі архітектурні рішення та варіанти розміщення застосунку.

2.1. Огляд архітектури застосунку

2.1.1. Однорівнева архітектура

Однорівневі застосування є одним з найпростіших типів архітектури. Суть такого підходу полягає в тому, що дані зберігаються на одній машині з клієнтським застосунком і виконуються в одному процесі. Клієнт робить запити до серверу бази даних напряду через мову SQL. Це дає змогу використовувати застосунок без доступу до мережі, адже

зв'язок між його частинами використовується на локальній машині.

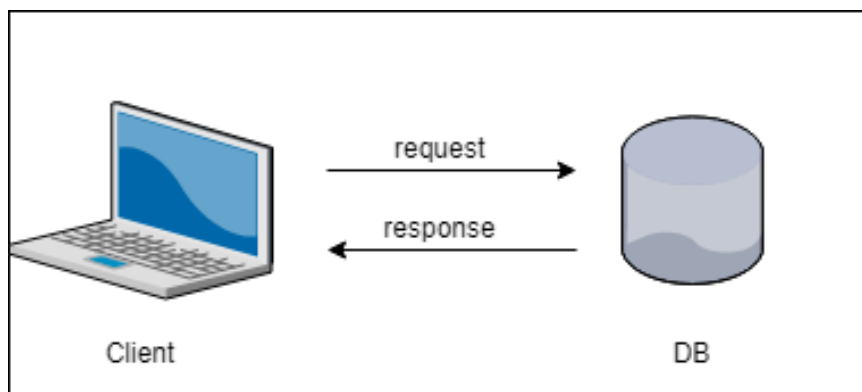


Рисунок 1 - модель однорівневої архітектури

2.1.2. Дворівневі застосування

У дворівневого застосування додається сервер, який напряму відповідає на запити клієнта і використовує свої ресурси для обробки запитів. Сервер обробляє дані, які отримує від бази даних та передає їх клієнту. Такий підхід дозволяє збільшити надійність застосунку та покращити його масштабованість. В залежності від розміщення компонентів, на стороні клієнта чи серверу, визначається основна модель взаємодії в рамках дворівневої архітектури.

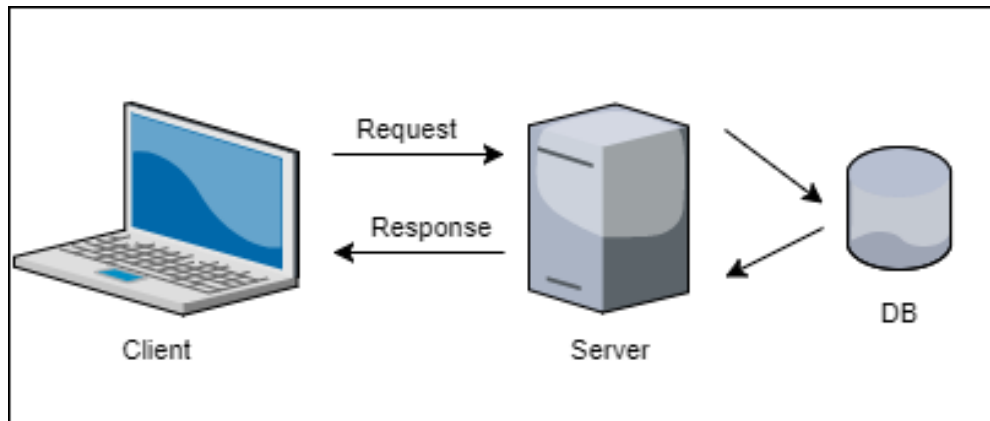


Рисунок 2 - модель дворівневої архітектури

2.1.3. Три-рівневі застосування

Три-рівневі застосування відрізняються тим, що між клієнтом та сервером використовується проміжний рівень, який називається сервером застосунку (application server), в якому описується бізнес-логіка застосунку. Сервер застосунку обробляє дані в необхідному форматі, які отримує від сервера, та передає їх клієнту. Наразі, три-рівнева архітектура є однією з найпопулярніших у розробці веб-застосунків

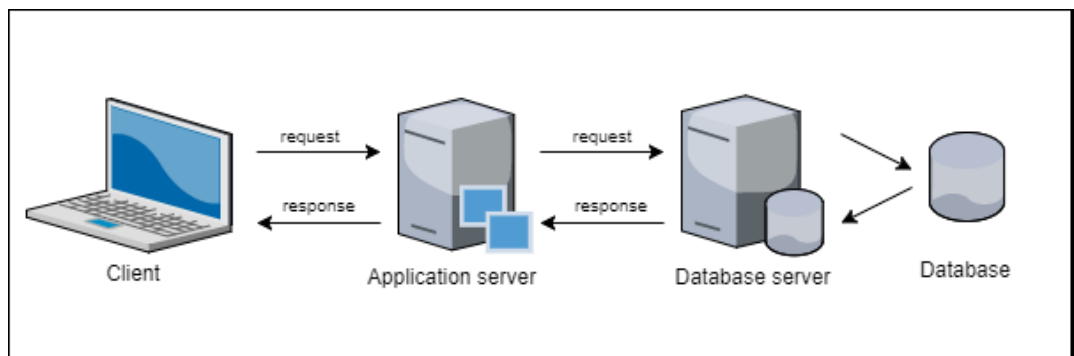
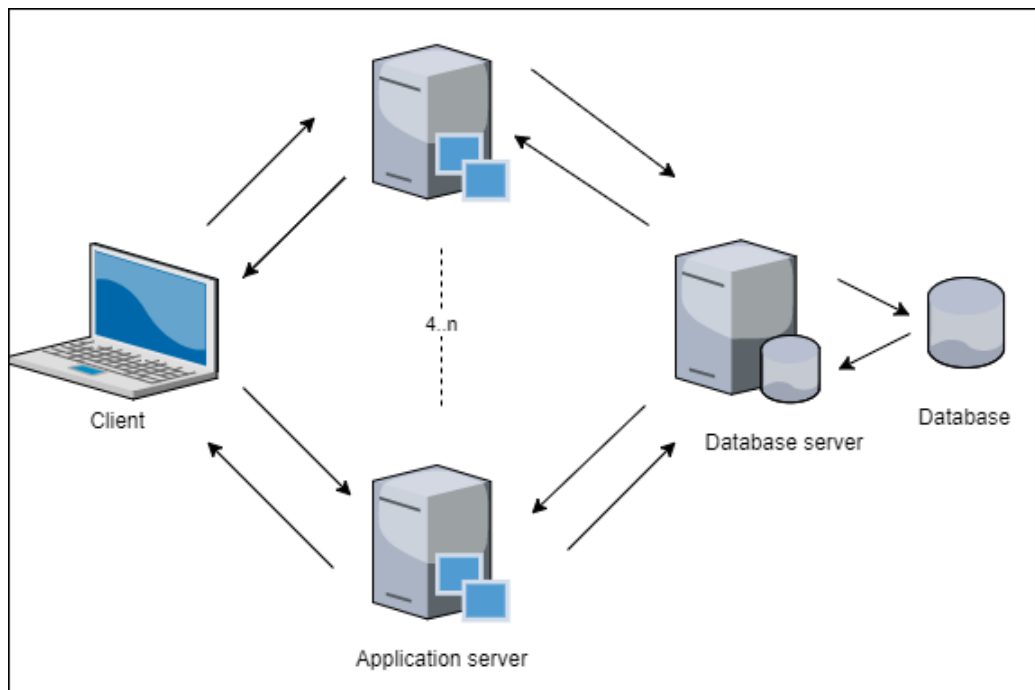


Рисунок 3 - три-рівнева архітектура застосування

2.1.4. Багаторівневі застосування

Багаторівнева архітектура застосунку – це розширена три-рівнева архітектура, шляхом додавання додаткових серверів застосунку та збільшенням специфікації кожного з них. Для досягнення 4,5,..., n-рівнів застосування виділяються додаткові рівні обробки даних, кожен з яких представляє власний сервіс та користується послугами інших серверів різного рівня.



2.1.5. Порівняльний аналіз

Проведемо порівняльний аналіз архітектурних рішень для веб-застосунку, в якому наведемо плюси та мінуси кожного вищезгаданого рішення.

Тип архітектури	+	-
1-рівнева архітектура	Можливість роботи без доступу до мережі Швидка у розробці та розгортанні	Низька масштабованість Низький рівень безпеки даних
2-рівнева архітектура	Можливість налаштовувати сервер бази даних Можливість розподілити навантаження між клієнтом та сервером	Можливі складнощі при забезпеченні безпеки даних Низька масштабованість
3-рівнева архітектура	Можливість забезпечити безпеку даних Можливість налаштувати спеціалізацію сервера бази даних Можливість масштабування застосунку	Складніше у розробці та підтримці кожного з рівнів архітектури Високі потреби в ефективній роботі серверів застосунку та сервера бази даних, що веде до великої вартості необхідного обладнання
4.. n – рівнева архітектура	Легко масштабувати Налаштування специфікацій серверів	Складно розробляти та підтримувати багаторівневу архітектуру застосунку Велика вартість через необхідність розміщувати дані на нових серверах при збільшенні масштабування

Для розробки розподіленого застосунку онлайн-магазину я планую використовувати багаторівневу архітектуру. Адже такий підхід дозволяє максимально розподілити навантаження

між частинами застосування та максимально збільшити продуктивність його роботи.

2.2 Огляд варіантів розміщення

2.2.1. Оренда дата-центру з розміщенням фізичного обладнання

Дата-центри – це спеціальні приміщення для розміщення мережного або серверного обладнання. Ви можете орендувати дата-центр під власне обладнання, або використати вже наявне обладнання дата-центру. Вони призначені для зберігання великих масивів інформації з великою швидкістю обробки та безпеки даних. Для кращої швидкості передачі даних, ви можете обрати дата-центр, який буде підлягати під усі ваші вимоги, починаючи від ціни і закінчуючи геопозицією. Дата-центри мають 4 критерії оцінювання обробки даних, вони називаються “tier”. Оренда дата-центру з розміщенням фізичного обладнання є гарним рішенням, якщо необхідно обробляти високі обсяги інформації з максимальною швидкістю та гарантованою безпекою вашого обладнання. Адже в дата-центрі є цілодобовий обслуговуючий персонал, який зможе усунути проблеми з сервером та відновити його працездатність. Також необхідним критерієм для підтримання

даних з найкращою відмовою стійкістю та високою доступністю, необхідна бути доступна кластеризація серверів. Це необхідно для того, щоб підтримувати стабільне навантаження та розподіляти його між серверами, навіть якщо один з них відпаде. Дата-центр надає послуги : оренда серверних шаф, оренда серверу, під'єднання до каналу зв'язку, віртуальний сервер, або віртуальний хостинг. Для вибору дата-центра з найкращим рівнем доступності, потрібно обирати лише ті, у яких відсоток доступності буде вищим за 99,95%. Такий варіант найкраще підійде для малого та середнього бізнесу для швидкого росту та масштабування великих обсягів даних.

2.2.2. Розміщення на власних ресурсах

Розміщення на власних ресурсах є гарним вибором для збереження даних для швидкого доступу до них та повного контролю ресурсів. Проте для використання такого рішення потрібно забезпечити цілодобову безпеку обладнання та приміщення для його зберігання. Підтримка свого дата-центру є доволі дорогою та трудомісткою. Адже на серверах потрібно постійно проводити технічну перевірку обладнання та встановленого програмного забезпечення. Такий варіант підійде для компаній, яким необхідний автономний доступ до даних навіть в час повної втрати зв'язку через локальну мережу, але потрібно бути готовим вирішувати проблеми з сервером в будь-який момент часу. Також необхідно розглянути питання

створення серверного кластера в своїй організації, адже це забезпечить кращу відмовостійкість та безпеку для ваших даних. Але це буде забирати велику кількість ресурсів на підтримку робочого кластера.

2.2.3. Розміщення на хмарних ресурсах

Розміщення на хмарних ресурсах є одним з найпопулярніших рішень для великої маси застосунків. Адже такий спосіб дозволяє вам адмініструвати свої ресурси в будь-який момент часу, надає великий вибір варіантів геопозиційного розміщення та регулювати вартість кожного розгорнутого ресурсу. Великою перевагою хмарних ресурсів полягає в тому, що ви платите лише тоді, коли користуєтеся. Це дозволяє уникнути величезних втрат коштів в той час, коли деякі з компонент не використовуються.

Також розміщення на хмарних ресурсах надає можливість автоматизувати процес оновлення застосунку через системи контролю версій, такі як Git.

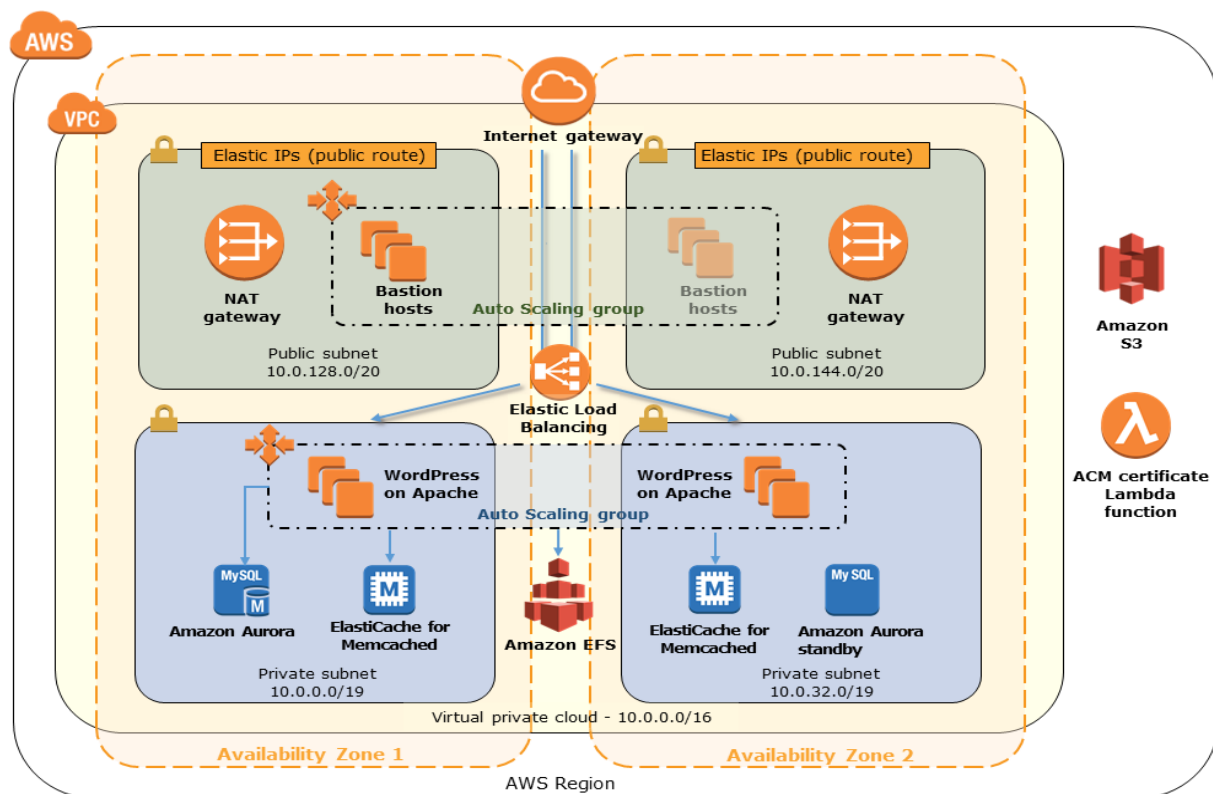
Хмарні технології на сьогоднішній день зростають з дуже великою швидкістю. Варіативність компаній, які готові надавати вам свої послуги зростають щосекунди. Найпопулярніші з них, це – AWS(Amazon), Azure(Microsoft), Google Cloud(Google). При виборі хостингу, ви можете обирати той, який буде найкраще підходити вам за потребами та ціною.

Послуги хмарних провайдерів можуть вам забезпечити майже 100% доступність до ваших ресурсів. Найчастіше, доступність вимірюється в відсотковому співвідношенні. Число дев'яток ідентифікує рівень високої доступності.

Так число, яке рівне 99,99 % буде означати, що сервіс може бути недоступний приблизно 52,6 хвилин протягом всього року. Наприклад, система сервісу зберігання даних, така як S3 від Amazon має відсоток доступності 99.999999999%.¹

¹ <https://cloud.netapp.com/blog/understanding-aws-high-availability-compute-sql-and-storage>

2.2.4. Порівняльний аналіз способів розміщення



Спосіб розміщення	+	-
Оренда дата-центру	<p>Підійде для швидкого росту та масштабування застосунку</p> <p>Цілодобовий доступ до даних</p> <p>Невелика ціна оренди, відносно утримання власного обладнання</p> <p>Збереження даних від втрат шляхом резервного копіювання</p>	<p>Не підходить для активного зростання обсягу даних через постійну необхідність в додаванні серверного обладнання</p> <p>В деяких дата-центрах можливі ремонтні роботи, що можуть зупинити роботу обладнання</p> <p>Не можливо досягти ~100% доступності даних</p>
Розміщення на власному	<p>Повний контроль над обладнанням та його обслуговуванням</p>	<p>Великі витрати на електроенергію та обслуговування серверних шаф</p>

фізичному обладнанні	<p>Можливість оновлювати апаратну частину серверної шафи</p> <p>Можливість швидко оновлювати програмну частину серверу</p> <p>Можливість отримати доступ до даних по локальній мережі</p> <p>Контроль над доступністю серверного обладнання</p> <p>Можливість зберігання конфіденційної інформації</p>	<p>Потрібно цілодобово відстежувати стан серверу для уникнення несправностей</p> <p>Потрібно виділяти окреме приміщення для серверного обладнання</p> <p>Необхідність створення серверного кластера для збереження даних в безпеці</p>
Розміщення на хмарних ресурсах	<p>Відносно низька вартість (платите тільки коли користуєтесь)</p> <p>Висока доступність (99,(9)%)</p> <p>Безпека даних, яка досягається децентралізованим зберіганням інформації</p>	<p>При великому споживанні ціна починає стрімко зростати</p> <p>Можливість непередбачуваних витрат при стрімкому зростанні трафіку</p> <p>Можливість втрати даних через проблеми провайдера</p>

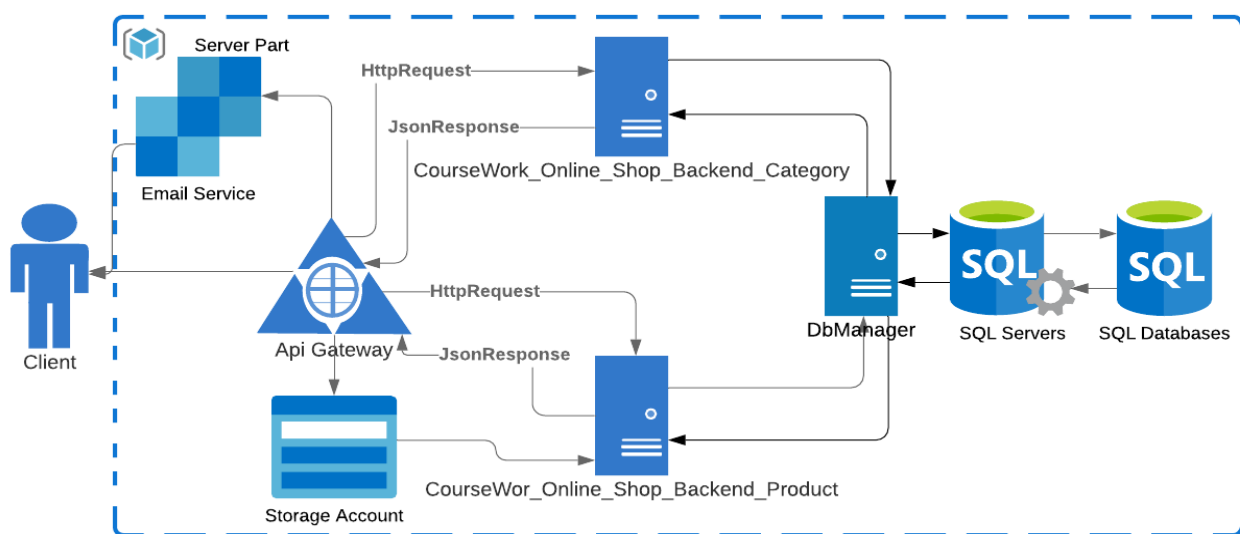
В своїй роботі я планую використати розміщення на хмарних ресурсах, через високий рівень доступності, низьку вартість та відсутність необхідності зберігати секретні дані користувачів. На хмарних ресурсах я зможу за малий проміжок часу розгорнути усі необхідні для роботи компоненти і одразу почати їх використання.

3. Розробка власного рішення

Для розробки власного рішення онлайн магазину, я вирішив обрати мову C# та крос-платформний фреймворк ASP.NET CORE для побудови клієнтської та серверної частини мого застосунку. Основними вимогами до застосування є :

- Зв'язок між компонентами має бути через Http запити;
- Дані мають зберігатися в реляційній базі даних;
- Приватна інформація про користувачів має бути в захищеному вигляді;
- Наявність користувацького інтерфейсу застосунку, щоб клієнт міг робити запити та переглядати наявні товари;

Для створення REST API для керування товарами та категоріями, було використано технологію Asp.Net Core Web Api, яка дозволяє налаштувати контролери для обробки інформації яка надходить від бази даних. Інтерфейс користувача виконує технологія Asp.Net Core Razor Pages, яка дозволяє легко обробляти користувацькі запити та спрощує побудову веб-застосування. Для отримання даних з бази даних я використав ORM Entity Framework Core, яка допомагає обробляти SQL запити та спрощує роботу з базою даних. В ролі бази даних використовується Azure Sql Db, а роль сервера бази даних виконує Azure Sql Server.



3.1 Структурна схема архітектури застосунку

Як можна бачити на структурній схемі, система має три пов'язані між собою компоненти. Це бізнес логіка застосунку та користувацький інтерфейс з визначеними зв'язками між ними. Компоненти контролю товарів та категорій є сервісами, які виконують REST архітектуру та мають виділений TCP порт для комунікації з іншими частинами застосунку. Ці компоненти також роблять запити до бази даних через ORM Entity Framework Core – інструмент реляційного відображення. Дані компоненти реалізують певну кількість ресурсів REST архітектури для опису дій, які можна зробити над об'єктом. Наприклад, метод GET який дозволяє отримати список усіх

об'єктів, або інформацію про об'єкт, який задовольняє певним параметрам. Метод POST дозволяє створити нову сутність об'єкта, або метод PUT, який дозволяє оновити дані про об'єкт. Також я використовую метод DELETE для видалення об'єкта з бази даних. Протокол передачі даних Http має декілька типів відповідей, де кожен тип має свій відповідний код. Це може бути інформаційне повідомлення з кодом – 1xx, або повідомлення про успішне виконання запиту – 2xx. Також є код перенаправлення – 3xx та коди про помилки : клієнтські помилки – 4xx або помилки сервера – 5xx. Дані у REST API передаються в серіалізованому форматі того формату даних, який буде вам зручніший. В залежності від ваших потреб ви можете передавати дані в форматі JSON, XML, тощо. Для передачі певного формату ви маєте встановити його в параметрі Content-Type. Наприклад, для формату JSON необхідно встановити значення “application/json”. В своєму продукті для серіалізації даних я використовую формат JSON, адже він зрозумілий для читання і легко може бути десеріалізований в необхідну сутність. Для зв'язування компонент між собою я використовую утиліту для керування контейнерами, яка називається Docker Compose. Це дозволяє мені об'єднати декілька контейнерів в робоче середовище та керувати ними. Контейнер керування продуктами працює на 5001 порті, а контейнер керування категоріями працює на 5002 порті.

Контейнер користувацького інтерфейсу працює на 80 порті та отримує дані від інших компонентів, на які він надсилає запит.

Компонент користувацького інтерфейсу розроблений з використанням технології Razor Pages на фреймворку Asp.Net Core. Діалог з бізнес-логікою відбувається через Http запити. Користувач може бачити тільки ті елементи, до яких він має доступ.

3.2 Опис компонентів застосунку

3.2.1. Компонент керування продуктами (CouseWork_Online_Shop_Backend_Product)

Даний компонент реалізує керування товарами в магазині. Для товарів реалізований контролер, який дозволяє переглядати список товарів, переглядати інформацію про обраний товар. Якщо ви створили товар, то ви можете видалити товар або змінити інформацію про нього. Даний компонент взаємодіє з базою даних, а саме таблицею ShopItems та маніпулює її даними. Компонент має зареєстровані шляхи, а саме :

- `api/product` – для отримання списку товарів (GET) або створенням нового товару (POST)

- `api/product/{id}` – для отримання інформації про певний товар (GET), оновленням інформації про певний товар (POST), або видалення певного товару (DELETE)

Всього даний контролер підтримує 4 методи, в залежності від обраної дії, ви отримаєте визначений результат. Даний компонент розроблений з використанням технології фреймворка Asp.Net Core Web Api, яка дозволяє створювати контролери, які будуть виконувати певну роботу при переході за певним шляхом.

3.2.2. Компонент керування категоріями (CourseWork_Onlile_Shop_Backend_Category)

Компонент керуванням категоріями дозволяє виконувати декілька операцій над категоріями товарів. В контролері реалізована можливість отримання списку усіх категорій; отримання інформації про певну категорію; створення, видалення або зміна інформації про певну категорію. Все це реалізовано за певними шляхами взаємодії, такими як :

- `api/category` – для отримання списку (GET) або створення нової категорії (POST)

- `api/category/{id}` – для отримання інформації про певну категорію (GET), видалення категорії (DELETE) або зміна даних про категорію (PUT)

До цього шляху може звертатись тільки користувач з правами адміністратора. Тільки він може керувати категоріями товарів на сайті. На етапі переходу до керування категоріями, перевіряється роль користувача. І якщо це не адмін, то він не може перейти до керування. Даний компонент також розроблений з використанням технології Asp.Net Core Web Api.

3.2.3 Компонент зв'язку з базою даних(DbManager)

За зв'язок із базою даних та отримання результатів, я використовую проект DbManager. Він надає можливість пов'язати сутності, які об'єднані в класі `AzureSqlDbContext` з сутностями, які знаходяться в базі даних. Це можливе завдяки використанню ORM Entity Framework Core, яка допомагає легко робити запити та діставати інформацію з бази даних. В цьому проекті визначається зв'язок з віддаленим сервером бази даних Azure Sql Db Server, який знаходиться в хмарному хостингу Azure та виконує роль сервера бази даних для хмарної бази даних Azure Sql Db.

3.2.4 Компонент користувацького інтерфейсу (CourseWork_Online_Shop_Frontend)

Даний компонент виконує більшу частину роботи застосунку, адже завдяки Razor Pages ви можете не лише створювати візуальну частину застосування, а ще обробляти запити, які створюються на певних сторінках. Для стилізації вигляду системи було використано bootstrap, який допоміг у використанні вже створених стилів для покращення зовнішнього вигляду. Компонент має сервіс реєстрації та логіну користувачів, можливість прочитати про політику користування сайтом або перейти на домашню сторінку. Також ви можете переглядати товари, або керувати категоріями (якщо ви є адміністратором сайту). Також є можливість переглядати дані про свій обліковий запис, або видалити себе з системи. На вікні товарів ви можете подивитись інформацію про вже існуючі товари, або придбати товари. Також ви можете створити новий товар для певної категорії, які існують на сайті.

3.2.4.1. Авторизація та аутентифікація

Авторизація та аутентифікація користувачів розроблена за допомогою Asp.Net Core Identity яка допомагає спростити процес авторизації користувачів у розробці. Інформація про користувачів зберігається в базі даних, а саме в таблиці AspNetUsers. Паролі користувачів зберігаються в зашифрованому вигляді з алгоритмом шифрування HMAC-SHA256 та надає можливість засекретити особистий обліковий запис. Також застосунку є можливість пройти аутентифікацію через сторонній сервіс, такий як : Microsoft Authentication, що дозволяє спростити етап реєстрації. Після реєстрації, користувач має підтвердити свою електронну адресу через електронний лист, який надійде одразу після реєстрації.

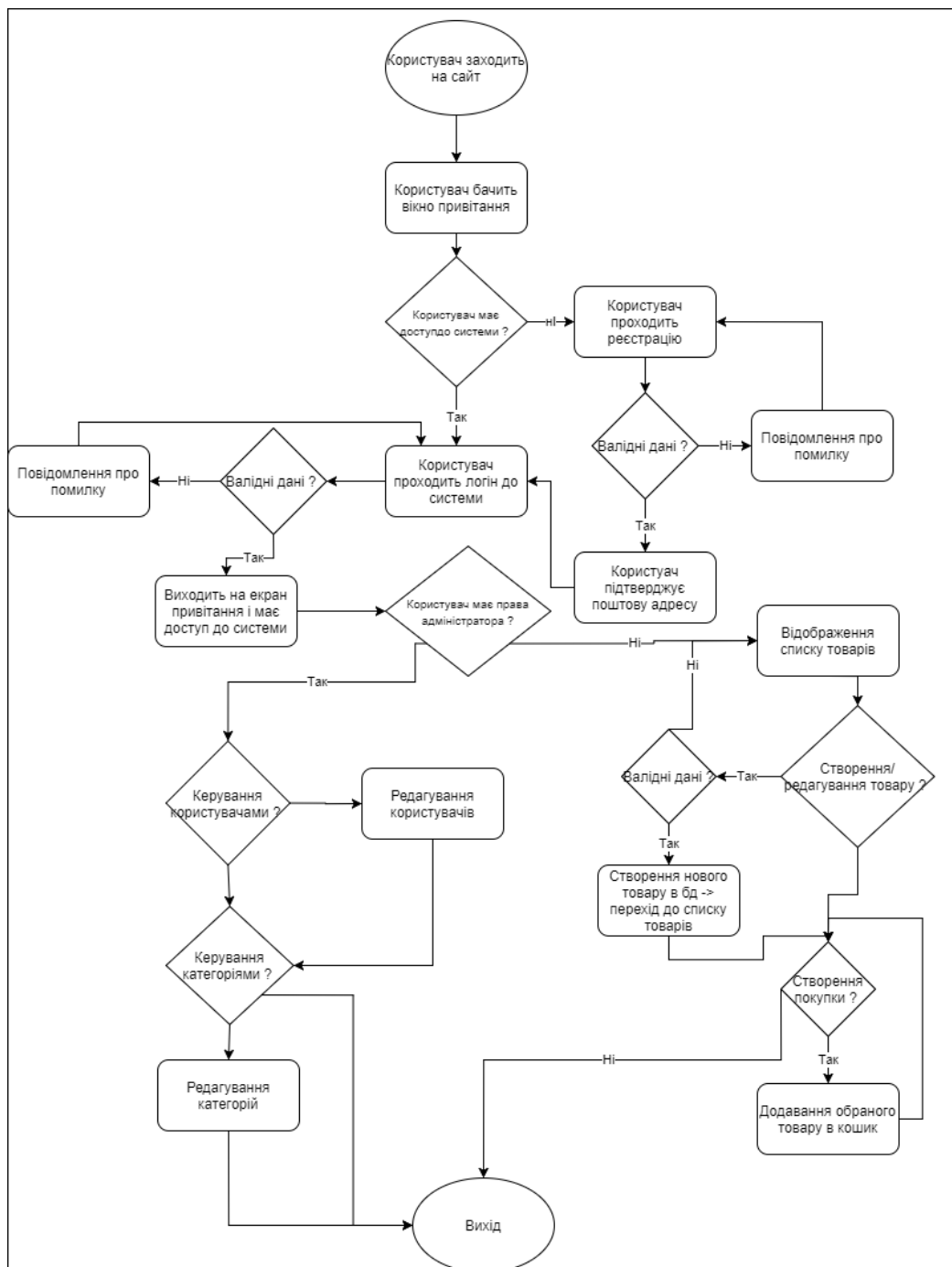
3.2.4.2. Сервіс керування користувачами

На сайті доступний сервіс керування користувачами. Якщо ви маєте права адміністратора, то ви можете перейти на даний сервіс та видаляти користувачів з системи. Це стирає сутність користувача з бази даних.

3.2.4.3. Сервіс створення покупки

Після того, як ви додасте необхідну кількість певних товарів у кошик, ви зможете зробити покупку, після чого вам надійде лист з інформацією про придбаний товар на електронну пошту. Для створення цього сервіса я використовував сесії, які з допомогою AntiForgery Token зробили можливим реалізацію кошика для певного користувача.

3.3 Опис функціонування системи



При переході на початкову сторінку, ви побачите головну сторінку з привітанням(Додаток 1), на якій буде можливість зареєструватися або увійти з вже створеним обліковим записом. Ви зможете зареєструватись або увійти в вже створеним обліковим записом на формі реєстрації або логіну (Додаток 2). Сторінка реєстрації містить 3 обов'язкових поля – це поштова адреса, яка буде використовуватись як логін користувача та пароль, який необхідно буде повторити. Після реєстрації ви маєте підтвердити свою пошту (на поштову скриньку надійде лист з посиланням для підтвердження), після чого вам необхідно буде залогінитись для користування сайтом. У випадку помилки на екран буде виведено повідомлення про помилку. Після аутентифікації, якщо ви маєте права адміністратора, ви зможете або перейти до вікна керування користувачами, або перейти до вікна керування категоріями (Додаток 3). Якщо користувач не має прав адміністратора, він може перейти до перегляду або керування товарами. На наступній сторінці для користувача відобразиться список товарів, в якому можна буде обрати деталі про товар, або купити товар(Додаток 4). Якщо користувач створив певний товар, то для нього відкривається можливість редагувати або видаляти цей товар. Користувач може додавати будь-яку кількість товарів у кошик(Додаток 5).

3.4 Розробка компонента користувацького інтерфейсу (CourseWork_Online_Shop_Frontend)

Для найпродуктивнішого процесу розробки необхідно підключити проект до системи контролю версій (наприклад : git). Це дозволяє контролювати зміни, які вносяться до проекту та переглядати зміни в файлах, що б не наштовхнутись на помилки. Для себе я налаштував сторінку репозиторію на GitHub, де налаштував собі дошку, яка за методологією SCRUM допомагає планувати хід робіт по проекту та дозволяє краще керувати часом виділеним на задачі. Також необхідно додати в кореневий каталог проекту файл .gitignore в який необхідно занести певні розширення, які не мають потрапити в загальний репозиторій (наприклад : ключі та параметри для підключення компонентів). Автоматично в репозиторії ви можете створити файл readme з коротким описом функціоналу або інструкцією по використанню продукту.

Для розробки компонента я обрав мову C# , а для зручності я обрав фреймворк для написання кросплатформених веб-проектів Asp.Net Core. Для встановлення необхідних залежностей я використовую вбудований менеджер пакетів NuGet PM, який дозволяє легко встановити будь-який пакет та знайти посилання на документацію. Файли залежностей

зберігаються в файлі проекту з розширенням .csproj . Файли проектів зберігаються в теці з файлом з розширенням .sln (його ще називають рішенням) яке зберігає метадані про проекти, які знаходяться всередині нього.

3.5 Загальна розробка веб-застосунку

Для розробки веб-застосунку було обрано фреймворк для побудови веб-проектів Asp.Net Core та спеціалізовані надбудови Web Api та Razor Pages. За допомогою цих надбудов ви отримуєте велику кількість засобів для побудови розподілених веб-сервісів які можна легко налаштувати під виконання необхідних завдань. За допомогою вбудованих пакетів для побудови REST архітектури, можна легко побудувати основний функціонал. Також гарним тоном при розробці буде винесення повторюваного коду в зовнішні сервіси та впровадження їх в проект через процес, який називається Dependency Injection, або впровадження залежностей. При створенні проекту ви можете обрати налаштування для https протоколу, що створить підписаний SSL сертифікат для використання захищеного зв'язку між компонентами.

За допомогою утиліти AspNet Core Identity можна створити початковий проект, який буде використовувати стилі bootstrap та дозволить легко налаштувати аутентифікацію користувачів.

Для створення нового веб-застосунку з використанням вбудованого інструменту Identity, необхідно в терміналі ввести таку послідовність команд : `dotnet new webapp --auth Individual` -о «назва застосунку». Після створення проекту автоматично створюється тимчасова база даних, яку можна змінити в будь-який момент. Для свого застосунку я обрав базу даних Azure Sql Db, яка знаходиться в хмарному хостингу Azure та дозволяє швидко та по захищеному каналу зв'язку отримати необхідні дані. Завдяки Azure я знаю що мої дані будуть в безпеці і я зможу отримати до них доступ майже в будь-який момент часу. В рішенні проекту ви можете додати ще велику кількість проектів, які вам будуть необхідні у розробці.

Для початку роботи із базою даних, необхідно обрати саму базу даних та сервер, який буде виконувати роль сервера бази даних. Я обрав хмарні рішення Azure Sql Db та Azure Sql Server через їхню надійність та легку взаємодію з проектами, які використовують платформу .NET. Після вибору бази даних, необхідно до проекту підключити Entity Framework Core, яка допоможе в керуванні та створенні нових сутностей у базі даних. Entity Framework Core – це ORM, яка розроблена компанією Microsoft та має великий функціонал для роботи з даними. Оскільки підключення до бази даних має бути захищеним, необхідно додати ці параметри в файл конфігурації застосунку `appsettings.json` . Також необхідно визначитись зі

сховищем фотографій товарів, адже при великій кількості користувачів зростатиме і кількість товарів в магазині. Для збереження медіа-даних я обрав Azure Blob Storage – це спеціалізоване сховище, в якому можна зберігати будь-які файли та при необхідності отримати їх по посиланню, яке надається кожному файлу при створенні.

Загалом, Asp.Net Razor Pages наслідує шаблон проектування MVVM (M-model, V-view, VM-viewmodel). Паттерн MVVM – це один з найбільш відомих паттернів для двосторонньої прив'язки даних. Завдяки Razor Pages можна легко розробляти веб-орієнтовані системи з дуже лаконічною структурою файлів.

Після визначення основних сутностей та зв'язків між ними, необхідно записати їх у клас DbContext для створення міграції. Міграції – це зміна схеми бази даних без втрати цілісності. Це означає, що ми можемо змінити структуру нашої бази даних в будь-який момент без втрат даних. Міграція визначає не лише схему створення сутностей бази даних, а й схему видалення. Кожна міграція впливає на дані, які вже знаходяться в базі даних. Завдяки Entity Framework Core ми можемо легко декількома командами створити нову міграцію. Після встановлення пакета Entity Framework Core та визначення класу DbContext, ми маємо додати сутності таблиць до цього класу і після цього в PackageManager Console виконати команду Enable-Migrations. Це дозволить зв'язати клас DbContext з

існуючою базою даних. Після того, як команда виконається успішно, необхідно виконати команду Add-Migration “Назва міграції”, це створить клас міграції, в якому будуть визначені зміни, які будуть внесені до бази даних. Після того, як будуть внесені остаточні зміни в файл міграції, необхідно виконати команду Update-Database, яка оновить модель бази даних та внесе визначені в класі міграції зміни. Для створення міграцій я використовую підхід Code-First. Суть цього підходу в тому, що потрібно спочатку кодом програми визначити сутності та зв’язки між ними, і після цього створювати міграції для оновлення бази даних.

Наступним кроком в розробці веб-застосунку я обрав створення початкової бізнес-логіки для визначення мінімального функціоналу, який має бути реалізований. Для створення бізнес-логіки я використав надбудову Web Api, яка дозволяє налаштувати роботу з базою даних через контролери API. Для використання цих контролерів, необхідно звернутись по певному шляху, та вказати вірний порт. Після цього контролер поверне дані в форматі JSON. Кожен контролер має свій сервіс, в якому визначені методи для роботи з базою даних. Ці сервіси(Додатки 6-7) можна підключити через наступний рівень абстракції – інтерфейс. Для використання цих сервісів через їхні інтерфейси необхідно зареєструвати їх в класі Startup.cs через механіку Dependency Injection.

Для розробки повного функціоналу необхідно додати логіку відображення даних для клієнта. Для цього в проєкті Razor Pages необхідно додати нове вікно, на якому будуть оброблятися запити клієнта до контролерів. Для цього необхідно визначити методи в моделі вікна (OnGet, OnPost, тощо). В залежності від дій користувача буде відображено або відповідь від сервера, або повідомлення про помилку. Для діалогу між компонентами я використовую Http запити, та передаю дані в форматі JSON. Для визначення, чи має користувач доступ до ресурсу, на початку кожного класу визначено атрибут Authorize, в якому визначено параметр Roles. Якщо користувач відноситься до ролі, яка визначена в атрибуті, то він зможе отримати до нього доступ. Якщо ні, то відобразиться помилка з кодом 401 Unauthorized.

Для серіалізації сутностей я використовую пакет Newtonsoft.Json, який дозволяє легко перетворювати сутності в об'єкти json та навпаки.

Одна із вбудованих особливостей Asp.Net Core Identity це створення зручного графічного інтерфейсу, який можна легко розширювати та використовувати вже вбудовані стилі, для розробки графічного інтерфейсу.

3.6 Розробка образу контейнера (Dockerfile)

Для побудови образу контейнера, в якому буде працювати ізолюваний компонент застосунку, необхідно розробити Dockerfile(додаток 7) в якому буде описано процес побудови образу контейнера. В цьому файлі ви можете описати послідовність певних команд та файл, який необхідно використати для побудови проєкту (файл з розширенням .srcproj).

На початку цього файлу необхідно вписати версію фреймворка, який використовується в проєкті та визначити директорію, в якій будуть зберігатись дані застосунку.

Наступним кроком необхідно визначити порт, який буде слухати застосунок, це можна зробити командою EXPOSE. Наприклад, проєкт CourseWork_Online_Shop_Frontend використовує для роботи 2 порти, це 80 для HTTP та 443 для HTTPS.

Наступним кроком в створенні образу, необхідно явно визначити робочу директорію та командою COPY визначити проєкти, від яких залежить проєкт, який буде зберігатись в образі контейнера. Команда COPY оновлює файли контейнера, які ще не зберігаються в ньому.

Для відновлення пакетів проекту, необхідно вписати команду `RUN dotnet restore "project_name"` це запустить команду відновлення залежностей.

Фінальним етапом в розробці докерфайлу, це визначення стеку виконання та бібліотеки, яка має бути запущена. Ця дія виконується командою `ENTRYPOINT`, де в квадратних дужках вказано платформу та назву бібліотеки. Наприклад, в проекті `CourseWork_Online_Shop_Frontend` це виконується таким чином: `ENTRYPOINT["dotnet","CourseWork_Online_Shop_Frontend.dll"]`.

Надалі образ контейнера використовується утилітою, яка дозволяє керувати та переглядати декілька контейнерів. Це утиліта `docker compose`. Далі в файлі `docker-compose.yml` визначається порядок та основні залежності в побудові образів контейнерів.

4. Висновки

В результаті моєї курсової роботи було розроблено веб застосування на основі багаторівневої мікросервісної архітектури з використанням контейнеризації.

За допомогою утиліти Docker Desktop можна переглядати та керувати існуючими Docker контейнерами. Застосунок розроблено з використанням великої кількості технологій : Asp.Net Core Razor Pages, Asp.Net Core Web Api, Azure Sql Server, Azure Sql Db, Event Grid, Azure Blob Storage, Bootstrap, Docker, Docker Desktop, docker-compose.

В своєму застосунку я розробив робочий прототип, який демонструє принципи багаторівневої архітектури та розподілених обчислень, яке складається з 3 компонентів, кожен з яких є загорнутим у контейнер.

В застосунку особлива увага приділяється полегшенню обчислень застосунку та збереження даних на сторонніх сервісах.

5. Список використаної літератури

1. Docker documentation - <https://docs.docker.com/>
2. Introduction to Razor Pages - <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio>
3. Configure multicontainer app with docker-compose - <https://docs.microsoft.com/en-us/visualstudio/containers/tutorial-multicontainer?view=vs-2019>
4. Introduction to High Availability - <https://cloud.netapp.com/blog/understanding-aws-high-availability-compute-sql-and-storage>
5. Azure Storage documentation - <https://docs.microsoft.com/en-us/azure/storage/>
6. Azure SQL documentation - <https://docs.microsoft.com/en-us/azure/sql-database/>
7. Docker compose overview - <https://docs.docker.com/compose/>
8. Razor Pages Pros And Cons - <https://stackify.com/asp-net-razor-pages-vs-mvc/>
9. Newtonsoft Json Serialization Documentation - <https://www.newtonsoft.com/json>

- 10. Asp.Net Core 3.1 documentation -
<https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1>
- 11. Інформація стосовно аренди дата-центру -
<https://freehost.com.ua/ukr/servers/datacenter/>
- 12. Configure local debugging with docker-compose -
<https://medium.com/it-dead-inside/docker-containers-and-localhost-cannot-assign-requested-address-6ac7bc0d042b>

6. Додатки

Додаток 1 (вікно привітання)

CourseWork_Online_Shop [Home](#) [Privacy](#)

[Register](#) [Login](#)

Welcome
You can now make some orders in CourseWork
Online Shop
Press any button to start using !

Додаток 2 (сторінка реєстрації/логін)

Register

Create a new account.

Email

Password

Confirm password

Register

Use another service to register.

Microsoft

Додаток 3 (вікно керування категоріями)

Index

[Create New](#)

Name	Description	
test	test description	Edit Details Delete
fruits	fresh fruits	Edit Details Delete

Log in

Use a local account to log in.

Email

larde2018@gmail.com

Password

.....

☐ Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)


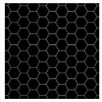

Use another service to log in.

Microsoft

Додаток 4 (список товарів)

Index

[Create New](#)


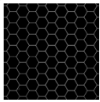
Name	Description	Price	Image	Category	
abc	abc	1		test	Details Buy
hexagons	just a lot of geometric hexagons	666		test	Edit Delete Details Buy
Huawei Mate 10	Simple Phone For Clever People	1000		test	Details Buy

Додаток 5 (Кошик)

[CourseWork_Online_Shop](#) [Home](#) [Privacy](#) [Items](#)

Hello ivish737@gmail.com! [Logout](#)

Cart

Action	Name	Image	Price	Quantity	<input type="button" value="Update"/>	Total
Delete	abc		1	<input type="text" value="1"/>		\$1
Delete	hexagons		666	<input type="text" value="3"/>		\$1998

Total \$1999

[Continue Shopping](#)

Додаток 6 (Сервіс обробки товарів)

ProductService.cs

```
namespace CourseWork_Online_Shop_Backend_Product.Services.ProductsService
{
    public class ProductService : IProductService
    {
        private IConfiguration _config;

        public ProductService(IConfiguration config)
        {
            _config = config;
        }

        public async Task<List<ShopItem>> GetAll()
        {
            var items = new List<ShopItem>();
            using(var db = new AzureSqlDbContext(_config))
            {
                items = await db.ShopItems.ToListAsync();
            }
            return items;
        }

        public async Task<ShopItem> GetById(Guid id)
        {
            var item = new ShopItem();
            using (var db = new AzureSqlDbContext(_config))
            {
                item = await db.ShopItems.Where(idProduct =>
idProduct.Id.Equals(id)).FirstOrDefaultAsync();
            }
            return item;
        }
    }
}
```

```

        public async Task<List<ShopItem>> GetProductsByCategory(string
category)
        {
            var items = new List<ShopItem>();
            using(var db = new AzureSqlDbContext(_config))
            {
                items = await db.ShopItems.Where(c =>
c.Category.Equals(category)).ToListAsync();
            }
            return items;
        }

        public async Task CreateProduct(string name, string description,
double price, string image, Guid categoryId, Guid userId)
        {
            using(var db = new AzureSqlDbContext(_config))
            {
                await db.ShopItems.AddAsync(new ShopItem(name, description,
price, image, categoryId, userId));
                await db.SaveChangesAsync();
            }
        }

        public async Task UpdateProduct(Guid productId, string name, string
description, double price, string image, Guid categoryId)
        {
            using(var db = new AzureSqlDbContext(_config))
            {
                var product = await db.ShopItems.Where(id =>
id.Id.Equals(productId)).FirstAsync();
                db.ShopItems.Update(product);
                product.Name = name;
                product.Description = description;
                product.Price = price;
                product.Image = image;
                product.CategoryId = categoryId;
            }
        }

```

```

        await db.SaveChangesAsync();
    }
}

public async Task DeleteProduct(Guid productId)
{
    using(var db = new AzureSqlDbContext(_config))
    {
        var product = await db.ShopItems.Where(id =>
id.Id.Equals(productId)).FirstAsync();
        db.ShopItems.Remove(product);
        await db.SaveChangesAsync();
    }
}
}
}

```


Додаток 7 (сервіс обробки категорій)

CategoryService.cs

```
namespace CourseWork_Online_Shop_Backend_Category.Services.CategoryService
{
    public class CategoryService : ICategoryService
    {
        private IConfiguration _config;

        public CategoryService(IConfiguration config)
        {
            _config = config;
        }

        public async Task<List<Category>> GetAll()
        {
            var categories = new List<Category>();
            using(var db = new AzureSqlDbContext(_config))
            {
                if(db.Categories != null)
                {
                    categories = await db.Categories.ToListAsync();
                }
            }
            return categories;
        }

        public async Task<Category> GetCategory(Guid? categoryId)
        {
            using (var db = new AzureSqlDbContext(_config))
            {
                var category = new Category();
                if(db.Categories != null)
                {
                    category = await db.Categories.Where(id =>
id.Id.Equals(categoryId)).FirstOrDefaultAsync();
                }
            }
        }
    }
}
```

```

        }
        return category;
    }
}

public async Task CreateCategory(string name, string description)
{
    var category = new Category(name, description);
    using(var db = new AzureSqlDbContext(_config))
    {
        if(!await db.Categories.ContainsAsync(category))
        {
            await db.Categories.AddAsync(category);
            await db.SaveChangesAsync();
        }
    }
}

public async Task UpdateCategory(Guid categoryId, string name, string
description)
{
    using(var db = new AzureSqlDbContext(_config))
    {
        var category = await db.Categories.Where(id =>
id.Id.Equals(categoryId)).FirstAsync();
        db.Categories.Update(category);

        category.Name = name;
        category.Description = description;

        await db.SaveChangesAsync();
    }
}

public async Task DeleteCategory(Guid categoryId)
{
    using(var db = new AzureSqlDbContext(_config))

```

```
        {
            var category = await db.Categories.Where(id =>
id.Id.Equals(categoryId)).FirstAsync();
            db.Categories.Remove(category);
            await db.SaveChangesAsync();
        }
    }
}
```

Додаток 8 (Dockerfile)

```
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
WORKDIR /src
COPY ["CourseWork_Online_Shop_Frontend/CourseWork_Online_Shop_Frontend.csproj",
"CourseWork_Online_Shop_Frontend/"]
COPY ["DbManager/DbManager.csproj", "DbManager/"]
COPY ["CourseWork_Online_Shop_Backend_Category/CourseWork_Online_Shop_Backend_Category.csproj", "CourseWork_Online_Shop_Backend_Category/"]
COPY ["CourseWork_Online_Shop_Backend_Product/CourseWork_Online_Shop_Backend_Product.csproj", "CourseWork_Online_Shop_Backend_Product/"]
RUN dotnet restore "CourseWork_Online_Shop_Frontend/CourseWork_Online_Shop_Frontend.csproj"
COPY . .
WORKDIR "/src/CourseWork_Online_Shop_Frontend"
RUN dotnet build "CourseWork_Online_Shop_Frontend.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "CourseWork_Online_Shop_Frontend.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "CourseWork_Online_Shop_Frontend.dll"]
```

«___» _____ 2019 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студенту факультету _____ інформатики _____ спеціальності
«Комп'ютерні науки та інформаційні технології» Іщенко Івану
Олексійовичу

Вихідні дані:

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Анотація

Вступ

Розділ 1. Огляд існуючих рішень

Розділ 2. Структурна розробка власного рішення

Розділ 3. Детальна розробка компонента

Висновки

Додатки

Список використаної літератури

Дата видачі «___» _____ 2019 року

Керівник Черкасов Д. І. _____

(підпис)

Завдання отримав _____

Календарний план виконання роботи:

№ п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	20.11.2019	
2.	Ознайомлення з предметною областю.	21.11. 2019	
3.	Спілкування з куратором стосовно курсової роботи	21.12.2019	
4.	Пошук літератури та написання першого розділу	25.02.2020	
5.	Побудова структури веб-застосування (другий розділ)	14.03.2020	
6.	Розробка архітектури веб-застосування	10.04.2020	
7.	Початок виконання практичної частини	10.04.2020	
8.	Написання третього розділу	15.04.2020	
9.	Аналіз виконаної роботи з керівником	07.05.2020	
10.	Створення слайдів презентації та написання тексту доповіді	09.05.2020	
11.	Корегування роботи	10.05.2020	
12.	Здача роботи на перевірку на плагіат.	11.05.2020	
13.	Захист курсової роботи	18.05.2020 – 28.05.2020	

Студент **Іщенко І.О.**

Керівник **Черкасов Д. І.** “_____” _____