МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Побудова багаторівневого веб-застосування на базі хмарної платформи Azure

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 4 курсу
Іщенко І.О.
"____" _____ 2020 р.

Зміст

1. Вступ 4

2. Огляд існуючих рішень 7

2.1 Зберігання даних 9

2.1.1. Azure Cosmos DB 9

2.1.2. Azure Storage 10

2.1.3. Azure SQL DB 11

2.2 Хостинг 12

2.3 Безпека 13

2.4 Обчислювальні ресурси 13

2.5. Порівняльний аналіз 15

3. Розробка власного рішення 16

Структурна схема архітектури застосунку 17

3.1. Опис компонентів застосунку 21

3.1.1. Компонент керування сутностями бази даних (GameLib_Back) 21

3.1.2 Компонент зв'язку з базою даних (DbManager) 23

3.1.3. Компонент користувацького інтерфейсу та бізнес логіки (GameLib_Front) . 23

3.2. Опис функціонування системи 26

3.3. Розробка компонента користувацького інтерфейсу 27

3.4. Загальна розробка веб-застосунку 28

3.5. Налаштування хмарної інфраструктури 30

4. Висновок 32

5. Список використаної літератури 33

6. Додатки 34

Додаток 1 – Блок-схема функціонування застосунку 34

Додаток 2 – Головна сторінка застосунку 35

Додаток 3 – сторінка перегляду відеоігор 35

Додаток 4 – особистий кабінет користувача 36

Додаток 5 – сторінка ігор для користувача в ролі адміністратора.....*36*
Додаток 6 – сторінка керування категоріями *37*
Додаток 7 – сторінка керування користувачами *37*
Додаток 8 – хмарна інфраструктура Azure.....*38*
Додаток 9 – контролер керування іграми.....*38*
Додаток 10 – сервіс керування іграми.....*40*

1. Вступ

Кожного дня користувачі світової мережі інтернет витрачають свій час для пошуку необхідної інформації. Задля полегшення пошуку, а також продуктивності використання власного часу, набирають популярності тематичні on-line бібліотеки. Такі ресурси допомагають отримувати більш розгорнуту інформацію, оглядати схожі варіанти, а іноді навіть купувати шукані товари. Кожна on-line бібліотека повинна підпадати під певні критерії, для того, щоб клієнти були задоволені використанням продукту :

- збереження даних у захищеному сховищі;
- відмовостійкість;
- динамічна масштабованість;
- захист від зовнішніх загроз;

Для виконання цих критеріїв необхідно розмістити застосунок на захищеній інфраструктурі, яка надасть змогу працювати 365 днів на рік без перебоїв. Розміщення на власних ресурсах чи в орендованому дата-центрі може коштувати дуже великих грошей та часу на налаштування, а також необхідність постійної підтримки, що буде тільки затримувати розвиток бізнесу. Тому найбільш підходящим варіантом буде розміщення на хмарних ресурсах, що не тільки зменшить витрати на створення початкової

інфраструктури, а і дозволить розгорнути застосунок в будь-якій точці світу за лічені секунди. На сьогоднішній день у світі існує велика кількість хмарних хостингів, які надають широкий спектр можливостей для масштабування та розробки on-line застосунків. Кожен провайдер має свої особливості, а також ресурси, які можуть задовольнити потреби бізнесу. Клієнти можуть використовувати 4 типи побудови хмарної інфраструктури, використовуючи різні типи хмар :

- Публічна хмара (побудова інфраструктури використовуючи ресурси провайдера хмарних послуг);
- Приватна хмара (побудова інфраструктури з використанням власних ресурсів);
- Гібридна хмара (побудова інфраструктури з використанням публічної та приватної хмари);
- Мульти хмара (побудова інфраструктури з використанням декількох типів хмар);

На сьогоднішній день побудова інфраструктури для розгортання застосунків на базі хмарних ресурсів, є одним із основних напрямків веб розробки та є дуже привабливою для розвитку та автоматизації процесів бізнесу.

Дана робота присвячена розробці on-line бібліотеки відеоігор на хмарній платформі Azure. Обраний провайдер хмарних послуг є одним із найкращих у світі і надає великий обсяг функціоналу із приблизно 600 ресурсів. Azure займає друге місце з популярністю серед провайдерів хмарних послуг та має найбільшу в світі кількість обчислювальних центрів. Microsoft Azure є надійним та безпечним провайдером хмарних послуг, адже вони використовують найновіші способи захисту даних та несуть відповідальність за нанесені втрати роботи бізнесу. Завдяки системі білінгу “pay-as-you-go” клієнти платять тільки за час використання ресурсів, що дозволяє розгорнути роботу застосунку не витрачаючи великих коштів на створення власної інфраструктури.

Метою моєї роботи є демонстрування переваг розробки застосунку на базі платформи Azure, та використання найновіших можливостей хмарного хостингу.

Дана тема є однією із найбільш актуальних сьогодні, адже майже кожен застосунок використовує можливості хмарних сервісів. А вміння користуватись та розробляти застосунки на базі хмарних платформ є невід'ємними навичками сучасних програмістів.

Ціль моєї роботи полягає в розкритті наступного :

- розробка відмовостійкої on-line бібліотеки для витримування великих навантажень;
- демонстрація переваг хмарної платформи Azure у розробці;
- використання найновіших хмарних ресурсів для покращення роботи застосунку;
- налаштування CI/CD для полегшення процесу оновлення функціоналу для неперервної роботи застосунку;

2. Огляд існуючих рішень

Розгортання онлайн бібліотеки на хмарній платформі може значно полегшити початкове налаштування застосунку. Використання хмарних ресурсів значно полегшує початкове розгортання та запуск застосунку. Використовуючи інструментарій провайдера хмарних послуг, ви можете створити ізольовану мережу і налаштувати відмовостійку систему, що дозволить забезпечити максимально надійну роботу застосунку. Найчастіше провайдери хмарних послуг мають сервіси, які дозволяють використовувати обчислювальні рішення, які необхідні виходячи із ваших потреб :

- IaaS (Infrastracture as a Service) – дозволяє розгортати інфраструктуру обчислювальних сервісів на базі дата-центрів провайдера;
- PaaS (Platform as a Service) – дозволяє отримати усі переваги IaaS, а також базову інфраструктуру. Провайдер самостійно керує елементами інфраструктури;
- SaaS (Software as a Service) – провайдер розміщує керує та доставляє усю інфраструктуру, а користувачі входять в систему для отримання доступу до ресурсів, які представляють певне рішення (наприклад, інструменти резервного копіювання та відновлення ресурсів).

Зазвичай, провайдери хмарних послуг мають різні сервіс, які розрізняються показниками відмовостійкості, ефективності роботи та вартості використання. Azure має багато суміжних сервісів, які мають схожий функціонал, але краще підходять для розв’язання певних задач.

2.1 Зберігання даних

Microsoft Azure має велику кількість сервісів для зберігання та обробки даних, кожен з яких має свої особливості та функціонал, які краще підійдуть для вирішення певних задач.

Основні критерії для вибору сховища даних :

- Формат даних;
- Розмір даних;
- Масштаб та структура;
- Зв’язки між даними;
- Гнучкість схеми;
- Життєвий цикл даних;
- Ефективність т масштабованість;
- Надійність;

Якщо правильно обрати рішення для зберігання даних, то можна підвищити ефективність роботи системи та зменшити витрати. В цьому розділі буде розглянуто основні сервіси для зберігання даних, які найкраще підходять для зберігання певних типів інформації.

2.1.1. Azure Cosmos DB

Azure Cosmos DB підтримує частково структуровані дані (NoSQL), що дозволяє змінювати структуру даних під час роботи,

задля збереження нової необхідної інформації не змінюючи основну модель. Кожна сутність індексується за замовченням, що дозволяє використовувати мову SQL для побудови запитів до бази даних.

Azure Cosmos DB підтримує гео-реплікацію, що дозволяє за один клік скопіювати дані в будь-яку точку світу щоб розширити доступність інформації для клієнтів з інших частин планети. Azure Cosmos DB має швидкість відклику менше 10 мілісекунд та рівень доступності 99,999%, а також інтерфейс API для MongoDB та Cassandra. Також база даних підтримує автоматичне масштабування при збільшенні навантаження на систему, що дозволяє покращити показники ефективності системи.

2.1.2. Azure Storage

Часто для бізнесу є критичним зберігати великі об’єкти такі як фото або відео файли. Такі дані доволі важко зберігати в звичних базах даних, тому що такі файли перевантажують систему і затримують час роботи запитів.

Azure Storage дозволяє зберігання великих двійкових об’єктів, зберігаючи при цьому швидкодію відповіді на запити. Дане сховище має дуже великий рівень доступності на рівні шістнадцяти дев'яток і підтримує гео-реплікацію з гнучким

масштабуванням ресурсів, що дозволяє зберігати та отримувати дані в будь-який момент. Використовуючи Azure Storage можливо змінювати рівні даних між «hot» та «cool», що дозволяє зменшити витрати на зберігання та збільшення пропускної здатності для найбільш використовуваних об’єктів.

2.1.3. Azure SQL DB

База даних Azure SQL DB представляє собою інтелектуальну та масштабуєму службу реляційної бази даних, спеціально оптимізовану для роботи в хмарі. Завдяки безсерверним обчисленням та hyperscale варіантом сховища, можливо автоматично масштабувати ресурси для того, щоб зосередитись на розробці продукту, не переживаючи за об’єм ресурсів. Azure SQL DB має багаторівневий захист та інтелектуальну систему пошуку загроз, які дозволяють забезпечити безпеку даних.

Високий рівень доступності даних на рівні 99.995% часу на рік, дозволяє будувати надійну систему, з якою зберігаєма інформація буде завжди готова за запитом.

Гіпер масштабування ресурсів бази даних дозволяє адаптуватися до змін, оперативно збільшуючи розмір сховища аж до 100 ТБ, а гнучка хмарна архітектура дозволяє збільшувати розмір за необхідністю.

Також база даних дозволяє створювати резервні копії за допомогою яких можна відновити сервіс за раховані хвилини незалежно від розміру операції.

2.2 Хостинг

Для хостингу веб-застосунків Microsoft Azure пропонує Azure App Service.

Azure App Service – це служба на базі протоколу HTTP для розміщення веб-застосунків та інтерфейсів REST API. App Service дозволяє розгортати застосунки на базі ОС Windows або Linux.

Azure App Service має не тільки функціонал хостингу застосунків, а також функціонал з забезпечення безпеки, балансувальники навантаження а також інструментарій з автоматичного масштабування ресурсів. Також можна налаштувати пайплайн CI/CD для автоматичного внесення оновлень застосунку.

Azure App Service дозволяє побудувати ефективну, безпечну та надійну архітектуру, адже даний сервіс може обробляти більше 40 мільярдів запитів на день маючи рівень доступності 99,95% часу на рік. А завдяки інструментарію геореплікації можливо розгортати копії веб-застосунку в найбільших осередках

користувачів, що дозволить зменшити затримку на обробку запитів клієнтів.

2.3 Безпека

Задля забезпечення безпеки усіх сервісів Azure рекомендується використовувати Azure Defender. Даний сервіс моніторить робочі навантаження, завдяки чому можливо протистояти загрозам, як підбір по протоколу віддаленого робочого стола (XDR) або SQL ін’єкціям.

Azure Defender виконує моніторинг усієї мережі і інформує про підозрілі спроби доступу до мережі, або про спробу завантаження шкідливого програмного забезпечення. Також даний сервіс має пропозиції по безпеці, які можуть забезпечити максимальний захист застосунку.

2.4 Обчислювальні ресурси

За кількість обчислювальних ресурсів для App Service відповідає App Service Plan. Даний сервіс дозволяє обрати необхідний обсяг ресурсів для роботи застосунку.

Задля економії ресурсів та коштів, можливо розгортати декілька застосунків на одному плані, порівно розподіляючи між ними ресурси.

Плани розділяються на 3 цінові категорії :

- Розподілене середовище виконання (застосунок виконується на тій самій віртуальній машині, що й застосунки інших клієнтів);
- Виділенні обчислення (застосунок виконується на окремій виділеній віртуальній машині Azure. Спільно використовують обчислювальні ресурси тільки ті застосунки, які знаходяться в одному App Service Plan);
- Ізольований (застосунок запускає виділенні віртуальні машини в виділених віртуальних мережах Azure щоб забезпечити мережеву ізоляцію для обчислень);

Кожен рівень також представляє підмножину функцій та сервісів, як :

- Особисті домени;
- Сертифікати TLS/SSL;
- Автоматичне масштабування;
- Слоти розгортки;
- Резервне копіювання

Azure App Service підтримує горизонтальне та вертикальне масштабування ресурсів. Вертикальне масштабування означає, що змінюється ефективність ресурсу. Це може бути

збільшення/зменшення розрміру віртуальної машини. Вертикальне масштабування часто виводить ресурс з роботи на деякий час, тому воно рідко виконується автоматично. Горизонтальне масштабування означає додавання чи виділення екземплярів ресурсів, які буде використовувати застосунок. Завдяки вбудованому балансувальнику навантажень при підготовці нових ресурсів застосунок буде працювати без перерв.

2.5. Порівняльний аналіз

Для розробки on-line бібліотеки відеоігор планується використати реляційну базу даних для збереження даних про ігри, разом із фото обкладинок. Оскільки зберігати фото в базі даних є доволі важкою задачею з точки зору використання ресурсів, тож найкраще буде використати Azure Storage для зберігання фото, а також Azure SQL DB як базу даних. Це дозволить побудувати надійну та ефективну систему, яка буде швидко відповідати на запити користувачів та безпечно зберігати дані.

Для хостингу бібліотеки планується використати Azure App Service що дозволить легко налаштувати процес розгортання застосунку, а також налаштувати інструменти моніторингу за

системою. App Service дозволяє налаштувати велику кількість інструментів всередині одного сервісу.

Задля забезпечення безпеки застосунку та особистих даних користувачів, планується використати Azure Defender, який дозволить побудувати безпечну інфраструктуру, а також отримати пропозиції з захисту.

Як план обчислювальних ресурсів планується використати розподілене середовище виконання для економії коштів, а також для демонстрації ефективності роботи сервісів Azure навіть на найдешевшому середовищі.

3. Розробка власного рішення

Для розробки on-line бібліотеки відеоігор, обрано мову C# та фреймворк Asp.Net Core для розробки крос-платформенного рішення для побудови клієнтської та серверної частини, а також для розробки клієнта взаємодії з базою даних використовуючи ORM Entity Framework Core. Основними вимогами до застосунку є :

- Використання протоколу HTTP для взаємодії між рівнями застосунку;

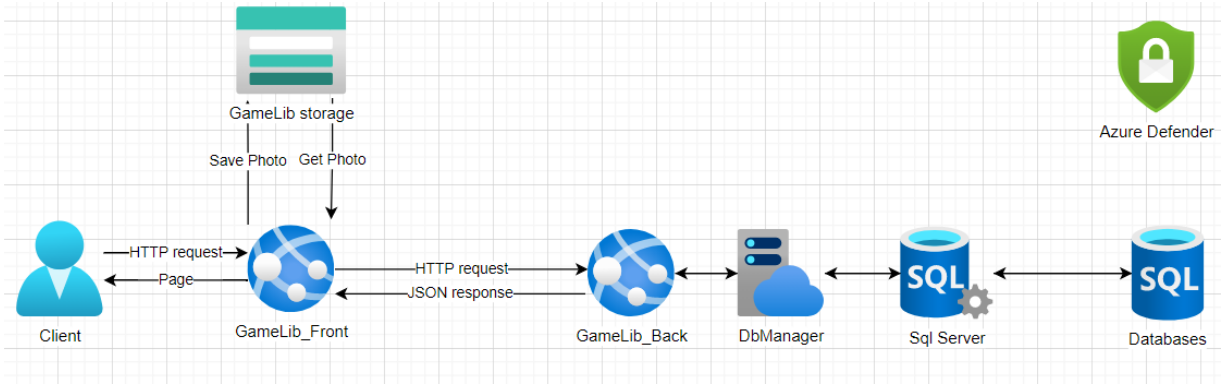
- Зберігання даних в реляційній БД;
- Наявність інтерфейсу системи для полегшення перегляду матеріалів бібліотеки;

Для розробки інтерфейсу REST API використано технологію Web Api на базі фреймворку Asp.Net Core для побудови контролерів API для обробки інформації, яка зберігається в базі даних. Web Api використовує HTTP протокол для обміну даними, що дозволяє використовувати його як веб сервіс для керування сутностями бази даних для зовнішніх сервісів.

Для побудови middleware та front частини застосунку використовується технологія Asp.Net Core Razor Pages яка дозволяє створювати обробники користувацьких запитів та розробляти користувацький інтерфейс використовуючи мову C#.

Для роботи з БД я обрав ORM (Object Relational Mapping) Entity Framework Core, яка дозволяє спілкуватися з БД не напряму, а через об’єкти .NET, що дозволяє захистити застосунок від SQL ін’єкцій. Для обробки запитів на стороні сервера бази даних використовується Azure Sql Server.

Структурна схема архітектури застосунку



За структурною схемою можна визначити чотири взаємопов’язані між собою компоненти, а також допоміжні хмарні сервіси. Цими компонентами є :

- Користувацький інтерфейс;
- Бізнес логіка застосунку;
- Клієнт для роботи з базою даних;
- Сервер бази даних;

Компонент GameLib_Back є частиною бізнес логіки застосунку і виконує взаємодію з сутностями бази даних через Entity Framework Core. GameLib_Back реалізовує архітектуру REST та має виділений TCP порт для обробки запитів з зовнішніх веб сервісів застосунку.

GameLib_Back реалізує такі методи REST архітектури, як :

- GET – для отримання списку існуючих об’єктів, або одного об’єкта за унікальним ідентифікатором;
- POST – для створення нового об’єкту;
- PUT – для оновлення існуючого об’єкту за унікальним ідентифікатором;
- DELETE – для видалення існуючого об’єкту за унікальним ідентифікатором;

Для відповідей на запити, які надходять на контролер, використовуються стандартні відповіді HTTP протоколу, які можна розрізнити за першою цифрою відповіді :

- 1xx – інформаційне повідомлення;
- 2xx – успішне виконання запиту;
- 3xx – перенаправлення на інший запит/сторінку;
- 4xx – помилка клієнта;
- 5xx – помилка сервера;

За REST архітектурою дані передаються в серіалізованому форматі даних, який є найбільш зручним. За замовчуванням REST архітектура використовує JSON формат даних.

Для розробки серверної частини on-line бібліотеки використовується JSON формат даних, адже він доволі зрозумілий

для усіх груп користувачів, не перевантажує систему, а також має велику кількість бібліотек для серіалізації об’єктів та десеріалізації JSON в об’єкти. Для спрощення операцій з JSON я використовую бібліотеку Newtonsot.Json яка є найпопулярнішою та простою в використанні.

Компонент користувацього інтерфейсу розроблений з використанням технології Razor Pages та фреймворк Asp.Net Core. Razor Pages дозволяє будувати більш прості та структуровані застосунки, адже код та логіку кожної сторінки описується в класі контролера, який прив’язаний до цієї сторінки. Єдиний контролер сторінки дозволяє поліпшити контроль застосунку та обробку помилок. Для зв’язку з бізнес логікою використовуються сервіси, які зареєстровані через механізм Dependency Injection, що дозволяє позбавитися прямих залежностей та керувати сервісами через абстракції. Сервіси створюють HTTP запити на певні контролери для керування сутностями БД.

В застосунку реалізовано рольову модель, щоб розділити доступ користувачів до певних частин інтерфейсу. Користувачі можуть бачити тільки ту частину користувацького інтерфейсу, до якої в них є доступ.

Для збереження фото ігор використовується надійне сховище, спеціалізоване для медіа файлів – Azure Blob Storage. Це дозволяє зберігати та отримувати зображення за створеним посиланням з швидкістю 500 запитів на секунду.

Для підтвердження реєстрації в застосунку та верифікації електронних поштових скриньок користувачів використовується інструмент SendGrid для відправки та обробки електронних повідомлень, а також збір статистики відповідей та унікальних кліків користувачів.

3.1. Опис компонентів застосунку

3.1.1. Компонент керування сутностями бази даних (GameLib_Back)

Даний компонент реалізує керування сутностями, які зберігаються в базі даних. Для керування даними реалізовані контролери, які :

- Дозволяють переглядати список сутностей, які зберігаються в базі даних;
- Переглядати інформацію про певну сутність за унікальним ідентифікатором;
- Створювати нову сутність;
- Оновлювати існуючу сутність за унікальним ідентифікатором;

- Видаляти існуючу сутність за унікальним ідентифікатором;

Даний компонент взаємодіє з базою даних через ORM Entity Framework Core, яка дозволяє керування сутностями бази даних через сутності .NET.

Компонент має контролери, для керування певними сутностями, а саме :

- CategoriesController – контролер для керування категоріями;
- GamesController – контролер для керування іграми;
- GenresController – контролер для керування жанрами;
- ModesController – контролер для керування режимами;
- PlatformsController – контролер для керування платформами;

Кожен з контролерів підтримує методи :

- GET /api/<controller name> - отримання списку сутностей;
- GET /api/<controller name>/id – отримання певної сутності;
- POST /api/<controller name> - створення нової сутності;
- PUT /api/<controller name>/id – оновлення певної сутності;
- DELETE /api/<controller>/id – видалення певної категорії;

Даний компонент розроблений з використанням технології Asp.Net Core Web Api, яка дозволяє реєструвати контролери за певними шляхами для виконання певних дій.

3.1.2 Компонент зв’язку з базою даних (DbManager)

Зв’язок з базою даних, а також визначення моделей сутностей та зв’язків між ними визначається в проекті DbManager за допомогою Entity Framework Core. В цьому проекті існує клас AzureSqlDbContext в якому визначені колекції DbSet<Model Name>, які пов’язані з таблицями в базі даних. Також використовуючи інструмент Fluent Api визначені зв’язки між сутностями, для побудови моделі в базі даних. Все це можливо завдяки використанню ORM Entity Framework Core, яка дозволяє робити запити до бази даних використовуючи код мовою C#. Сервером бази даних виступає Azure SQL Server, який бере на себе навантаження з обробки запитів до бази даних, що знижує навантаження на систему та пришвидшує виконання запитів. SQL Server зв’язує застосунок зі створеними базами даних Azure SQL DB, шлях до яких встановлюється в строках підключення.

3.1.3. Компонент користувацького інтерфейсу та бізнес логіки (GameLib_Front)

Компонент користувацького інтерфейсу містить основну частину бізнес логіки та більшу частину функціоналу застосунку.

Технологія Razor Pages дозволяє створювати веб сторінки та контролери для обробки запитів в суміжних класах, що дозволяє спростити написання коду та обробку помилок при запитах користувачів.

Для стилізації та полегшення розробки UI використовується bootstrap, який надає таблицю стилів та допомагає розробляти зручний інтерфейс.

Компонент має сторінку авторизації користувачів, особистий кабінет, сторінку перегляду ігор, можливо прочитати про політику зберігання персональної інформації. Якщо користувач має роль адміністратора, то він має доступ до сторінки керування іграми, сторінки керування категоріями, сторінки керування жанрами, сторінки керування режимами та сторінки керування платформами.

3.1.3.1. Авторизація та аутентифікація

Авторизація та аутентифікація користувачів реалізована з використанням темплейту Asp.Net Core Identity, яка надає можливість створення ефективної системи ідентифікації користувачів. Інформація про користувачів зберігається в окремій базі даних під назвою UserData. Усі паролі користувачів зберігаються в шифрованому вигляді, а важлива інформація

замаскована. Алгоритм шифрування HMAC-SHA256, яким захищені персональні дані, дозволяє надійно захистити особисті дані користувачів. Після реєстрації в застосунку, користувач має підтвердити свою електронну адресу через свою поштову скриньку, куди надійде лист з посиланням. Також в застосунку використовується сервіс Azure Active Directory який дозволяє ідентифікувати користувачів через аутентифікацію облікового запису Microsoft.

3.1.3.2. Керування користувачами

Керування користувачами дозволяє адміністратору застосунку видаляти користувачів, або змінювати їх роль. Сторінка доступна тільки для тих користувачів, які мають права адміністратора.

3.1.3.3. Персональний кабінет

Персональний кабінет користувача дозволяє керувати особистою інформацію, змінювати пошту, яка прив’язана до акаунту, змінювати пароль, а також завантажити дані, які збирає застосунок. Також користувач може видалити себе самостійно з застосунку.

3.1.3.4. Керування іграми

Звичайні користувачі системи можуть переглядати список усіх ігор бібліотеки, а також більш детальну інформацію про одну із

наявних ігор. Адміністратори системи можуть додавати нові ігри, а також керувати інформацією про вже існуючі. Також адміністратор може видаляти ігри.

3.1.3.5. Керування категоріями

Адміністратор системи може створювати та керувати категоріями, до яких потім буде додано нові ігри.

3.1.3.6. Керування жанрами

Адміністратор системи може створювати та керувати жанрами, які потім можна буде використовувати при створенні нових відеоігор.

3.1.3.7. Керування режимами

Адміністратор системи може створювати та керувати режимами, які потім можна буде використовувати при створенні нових відеоігор.

3.1.3.8. Керування платформами

Адміністратор системи може створювати та керувати платформами, які потім можна буде використовувати при створенні нових відеоігор.

3.2. Опис функціонування системи

В Додатку 1 визначено блок-схему функціонування застосунку для користувача. При переході за посиланням на основну сторінку, користувач побачить привітальне повідомлення

(Додаток 2), на якій буде можливість зареєструватися або увійти до системи. Авторизуватись у системі користувач може на сторінці реєстрації, або логіну, обравши зручний спосіб. Всього є два способи авторизуватись в системі :

- Створити обліковий запис;
- Обліковий запис Microsoft;

Після успішного логіну, користувач переходить на основну сторінку, звідки він може перейти до вкладки перегляду відеоігор (Додаток 3), або до особистого кабінету (Додаток 4). Якщо користувач має права адміністратора, то він також має доступ до функціоналу керування інформацією про відеоігри (Додаток 5), категорії (Додаток 6), жанри, платформи та режими. Також адміністратор має функціонал керування користувачами (Додаток 7), де він може змінювати їхню роль в системі, або видаляти їх.

3.3. Розробка компонента користувацького інтерфейсу

Для максимально ефективного процесу роботи, проект було підключено до системи контролю версій Git. Це надає змогу вносити зміни та зберігати їх в певному репозиторії, а також відновлювати проект, якщо виникнуть помилки. Також було налаштовано віддалений відкритий репозиторій на Github.

Для розробки даного компонента використовується фреймворк Asp.Net Core та технологію Razor Pages. Для установки необхідних пакетів використано вбудований менеджер пакетів для мови C# : NuGet Package Manager. NuGet має дуже зручну UI частину, яка полегшує пошук пакетів, обирання необхідної версії а також документацію до бібліотеки. Посилання на залежності зберігаються в файлі .cproj та перетворюються на .dll файли при збиранні проекту.

Для розробки користувацького інтерфейсу використовується інструментарій bootstrap, який надає базові шаблони CSS для покращення зовнішнього вигляду, не розробляючи власну таблицю стилів. В поєднанні з Razor Pages розробка UI не займає доволі багато часу.

3.4. Загальна розробка веб-застосунку

Початок створення веб-застосунку починається з вибору типу проекту, який буде розроблятись. Для створення веб-сервісу обробки HTTP запитів використано тип Web Api. Він дозволяє легко створювати та налаштовувати контролери для обробки запитів. А для створення веб-сервісу користувацького інтерфейсу використовується Razor Pages для побудови веб-сторінок швидко та ефективно. Для легшого початку створення застосунку, можна використати готові темплейти .NET, одним з яких є Identity, що

надає інструмент авторизації користувачів і легко налаштовувати під свої потреби. Для створення проекту з темплейту, необхідно обрати його через Visual Studio, або ввести команду в консоль, як : «dotnet new webapp». Після цього можна починати розробку проекту. Для розробки веб-застосунку необхідно сховище даних, яке буде доступним більше ніж 99% на рік та ефективним при великих навантаженнях. Azure SQL DB є гарним вибором, адже вона має доступність 99,995% часу на рік та має багаторівневий захист. Сервером бази даних виступає Azure SQL Server, яка має безсерверний рівень обчислень. Завдяки цьому, баз даних може масштабуватись залежно від навантажень. Після створення бази даних необхідно створити модель даних для майбутньої роботи. Asp.Net Core Identity надає базову модель даних для зберігання користувачів, а також налаштування рольової моделі. Внести зміни до бази даних можна через міграції. Міграції змінюють контекст бази даних зберігаючи при цьому історію змін. В проекті використовується підхід “Code First” в якому спочатку створюються моделі та зв’язки в коді, а потім міграціями вони змінюють структуру контексту бази даних. Для збереження фото відеоігор використовується сховище великих медіа об’єктів – Azure Blob Storage. Сховище спеціалізовано для збереження файлів та має SDK для .NET, яке спрощує завантаження та

отримання файлів. Сховище має найкращий показник відмовостійкості в Azure на рівні шістнадцяти дев'яток. Після того як початкову структуру створено, необхідно створити бізнес логіку застосунку. Для початку було створено Web Api проект, щоб визначити базову функціональність продукту. Контролери (Додаток 9) відповідають за роботу з певними сутностями. Для взаємодії з базами даних через абстракції підключено сервіси (Додаток 10) до контролерів, щоб позбавитись від прямої залежності, та надати гнучкості проекту. Кожен сервіс та інтерфейс необхідно зареєструвати в класі Startup через механізм Dependency Injection. Після створення бізнес логіки, необхідно створити інтерфейс для користувачів. Для цього необхідно створити нове вікно в проекті Razor Pages та налаштувати запити до сервера.

3.5. Налаштування хмарної інфраструктури

Для початку необхідно обрати географічну зону, яка буде основною для застосунку, адже це один з найважливіших параметрів. Обрання правильного регіону зменшить затримку запитів до бази даних та до застосунку від користувачів. Після вибору регіону необхідно створити групу ресурсів, в якій будуть знаходитись усі ресурси застосунку. Група ресурсів є логічним рівнем хмарного рішення, яке дозволяє відділити ресурси один від

одного за певним критерієм. Після створення групи ресурсів необхідно створити Azure Sql Server та базу даних, яка буде використовуватись у проекті. Для цього потрібно скопіювати строку підключення та зберегти значення в конфігурації проекту, для того щоб створити підключення до хмарної бази даних. Далі для хмарної інфраструктури необхідно створити сховище медіа файлів Azure Storage в якому налаштувати тип сховища, рівень обчислювальних потужностей та час зберігання об’єктів в режимі Hot. Після розробки продукту, необхідно налаштувати сервер, на який потім буде розгортатись застосунок. Azure App Service Plan відповідає за об’єм обчислювальних ресурсів та кількість додаткових сервісів, які покращують роботу застосунку. Чим кращий план – тим більша кількість ресурсів. Також є можливість створювати рівні середовища застосунку для миттєвого розгортання. Їх можна використовувати для тестування, розробки та для робочої версії. Після створення плану ресурсів, можна створювати сервіс для розгортання застосунку. App Service також має вбудовану функцію налаштування пайплайну CI/CD з найпопулярніших систем контролю версій таких як GitHub, BitBucket, а також Azure DevOps. Для цього необхідно обрати необхідний ресурс, ввести свої дані, а після цього обрати необхідний репозиторій. Azure автоматично налаштує пайплайн

та виконає розгортання в створений сервіс. Після цього необхідно перейти за посиланням, яке є в налаштованому сервісі, та перевірити чи все працює як потрібно.

4. Висновок

Результатом моєї курсової роботи є розроблене веб застосування на основі багаторівневої архітектури, яке використовує хмарну інфраструктуру Azure, та GitHub Workflow CI/CD.

За допомогою інструментів моніторингу на Azure можна слідкувати за станом застосунку та налаштовувати правила масштабування. Було використано велику кількість технологій для розробки : Asp.Net Core Razor Pages, Asp.Net Core Web Api, Entity Framework Core, Bootstrap, Dependency Injection. А також було використано хмарні сервіси, як : Azure Sql Server, Azure Sql Db, Azure Blob Storage, App Service Plan, App Service, Send Grid.

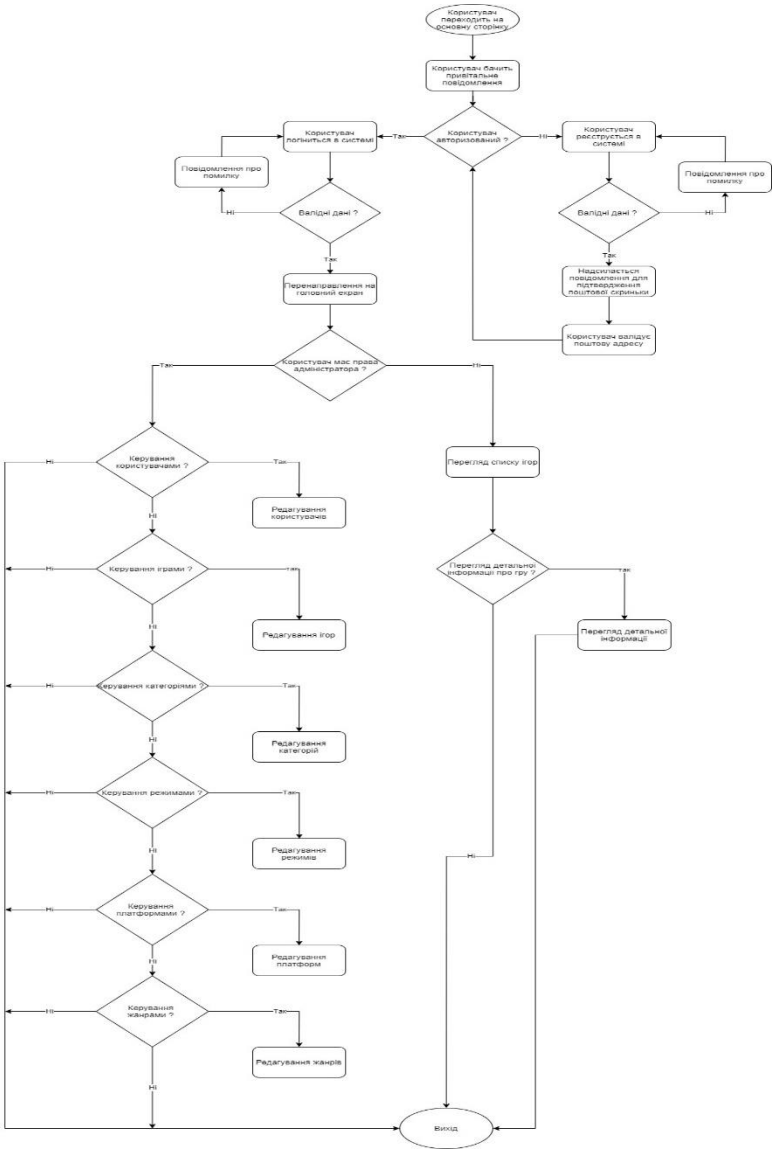
В своєму застосунку розроблено робочий прототип on-line бібліотеки, яка демонструє особливості сучасного підходу до розробки багаторівневих рішень. Застосунок складається з 3 робочих частин, кожна з яких функціонує в хмарі.

5. Список використаної літератури

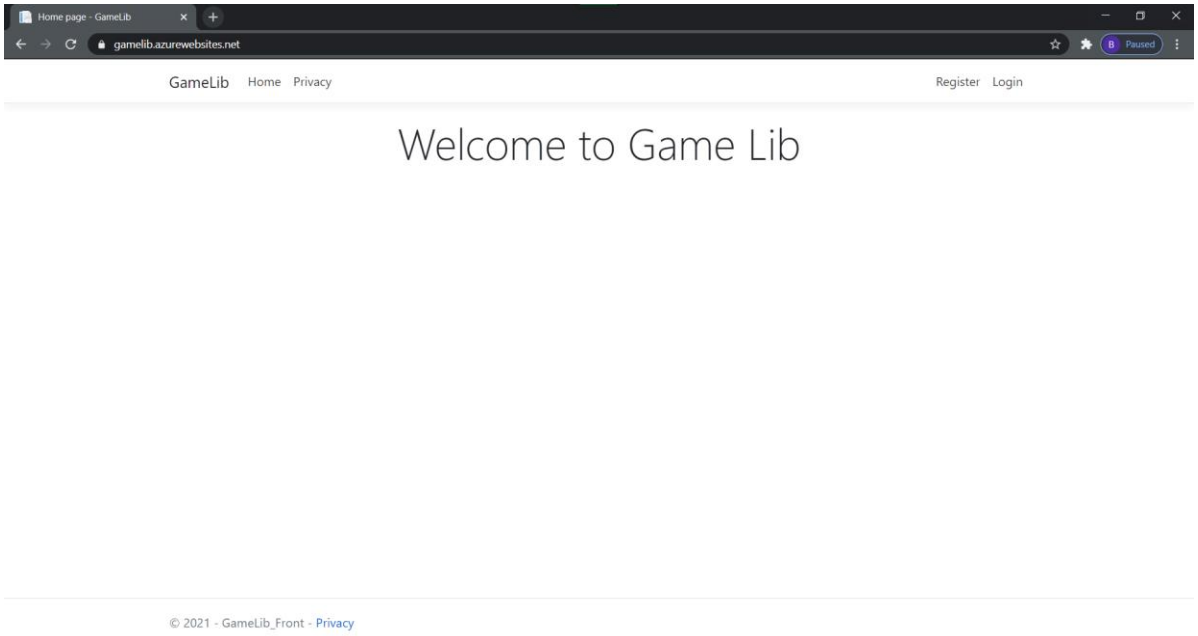
1. Azure Documentation - https://docs.microsoft.com/en-us/azure/?product=featured
2. Introduction to Razor Pages - https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio
3. Introduction to Web Api - https://www.tutorialsteacher.com/webapi/what-is-web-api#:~:text=The%20ASP.NET%20Web%20API,response%20instead%20of%20html%20view.
4. Azure Storage documentation - https://docs.microsoft.com/en-us/azure/storage/
5. Azure SQL documentation - https://docs.microsoft.com/en-us/azure/sql-database/
6. App Service Documentation - https://docs.microsoft.com/en-us/azure/app-service/overview
7. App Service Plan Documentation - https://docs.microsoft.com/en-us/azure/app-service/overview-hosting-plans
8. .Net Core 3.1 documentation - https://docs.microsoft.com/ru-ru/dotnet/fundamentals/
9. Remote debugging App Service in Visual Studio - https://docs.microsoft.com/en-us/visualstudio/debugger/remote-debugging?view=vs-2019

6. Додатки

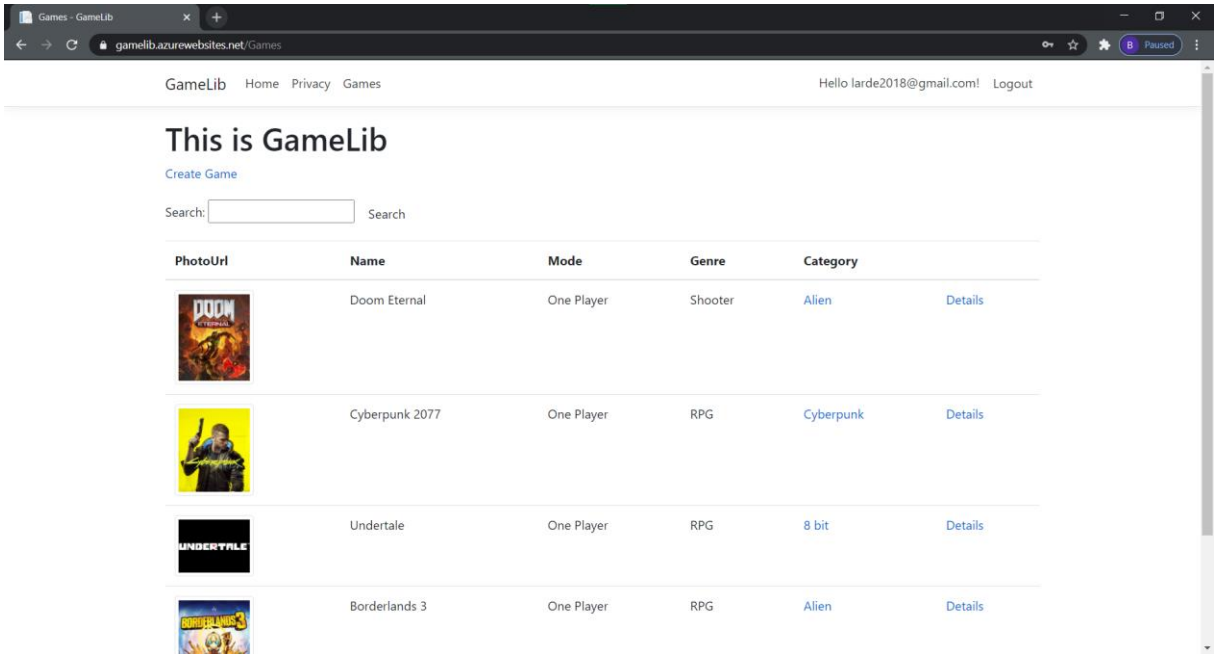
Додаток 1 – Блок-схема функціонування застосунку



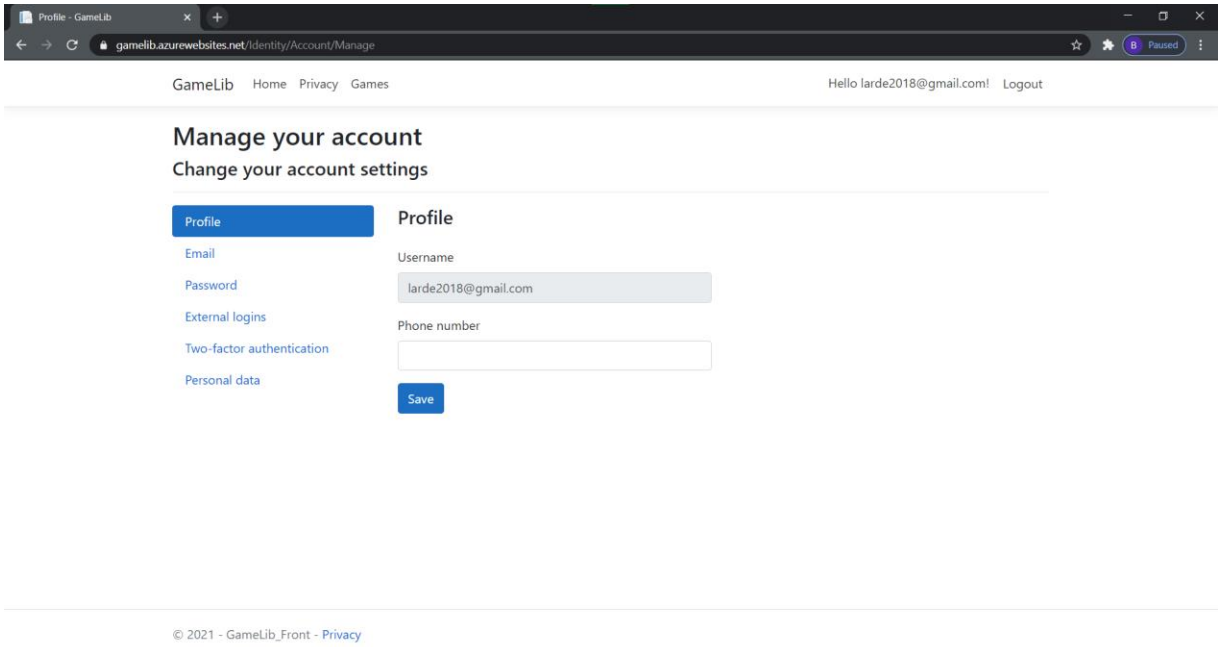
Додаток 2 – Головна сторінка застосунку



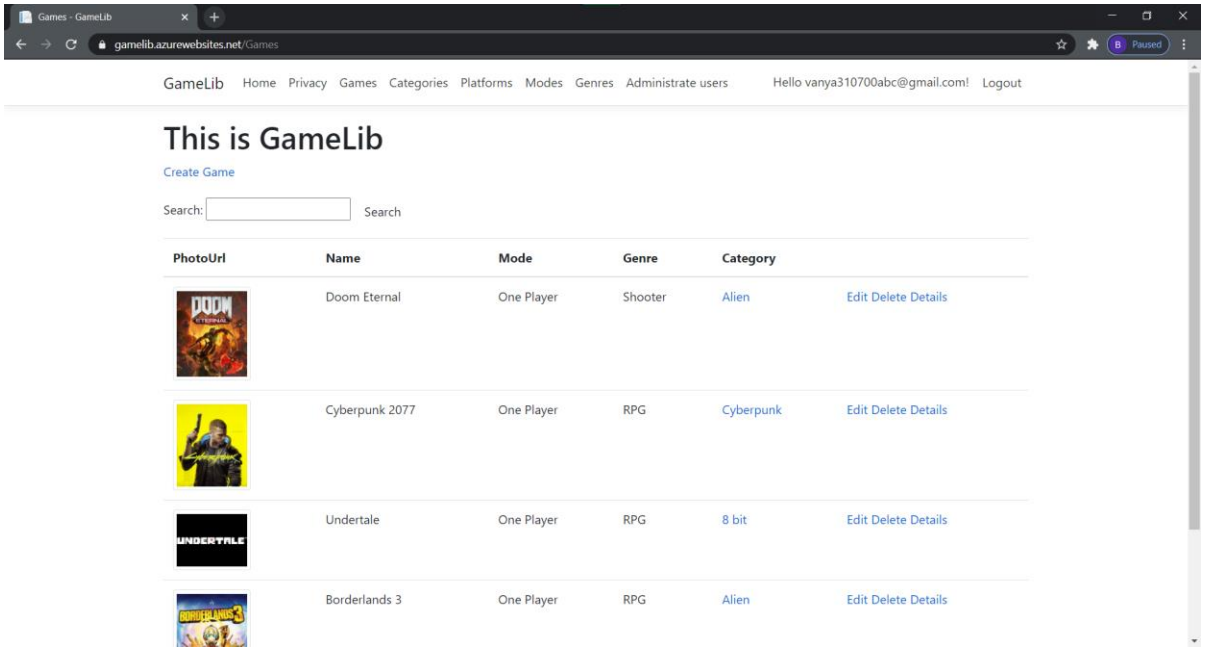
Додаток 3 – сторінка перегляду відеоігор



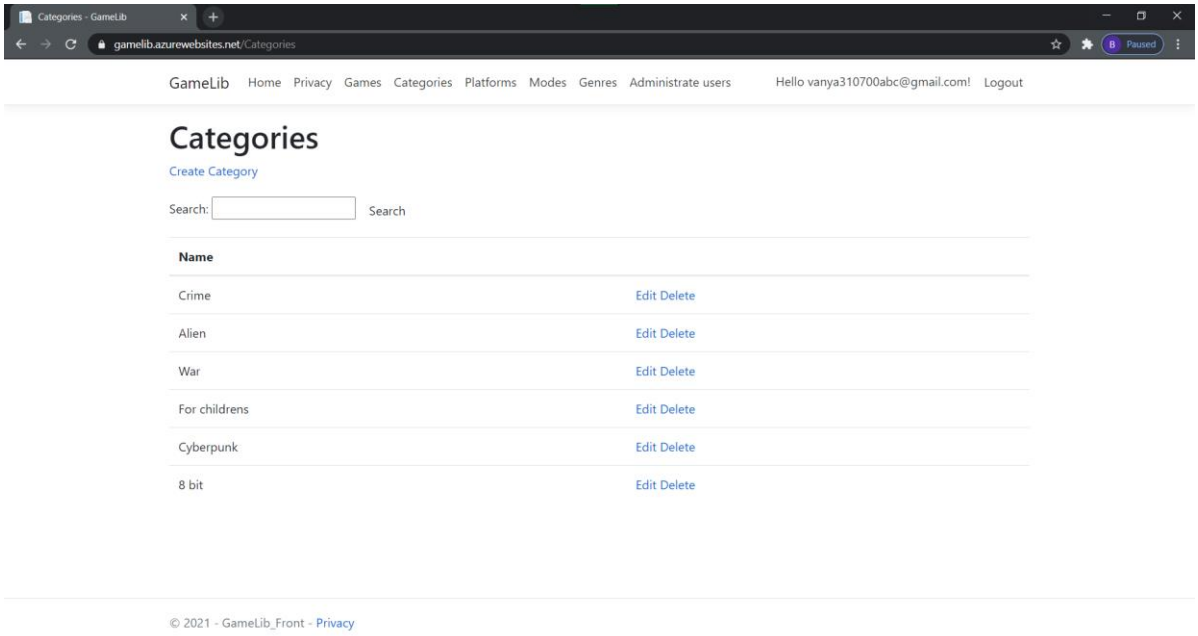
Додаток 4 – особистий кабінет користувача



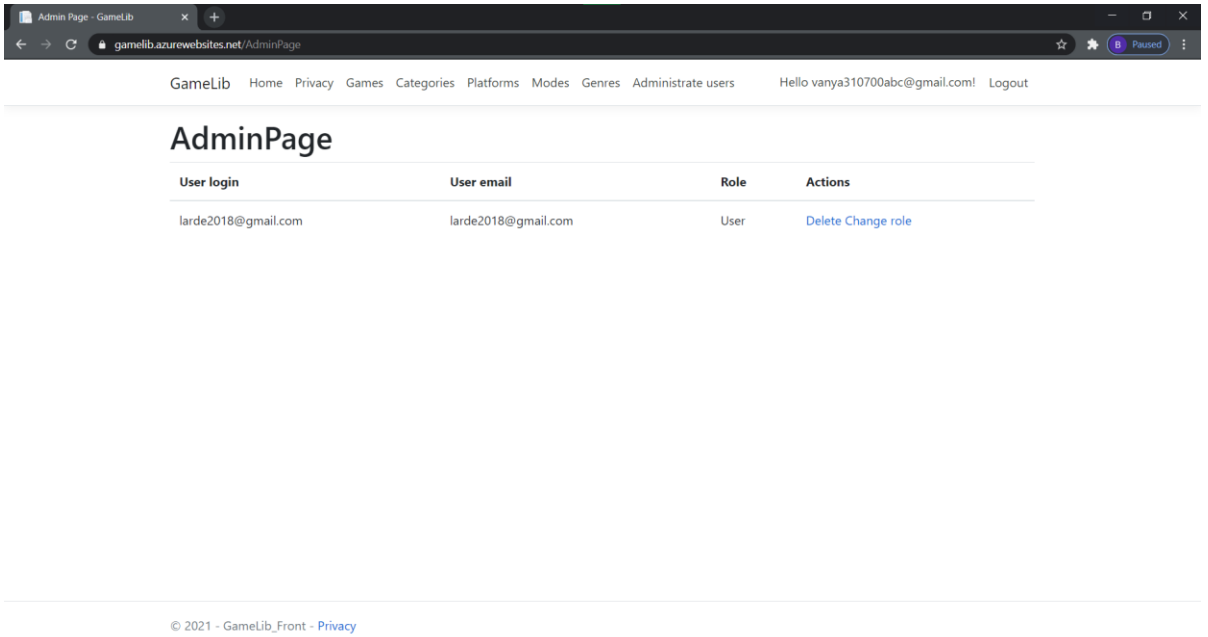
Додаток 5 – сторінка ігор для користувача в ролі адміністратора



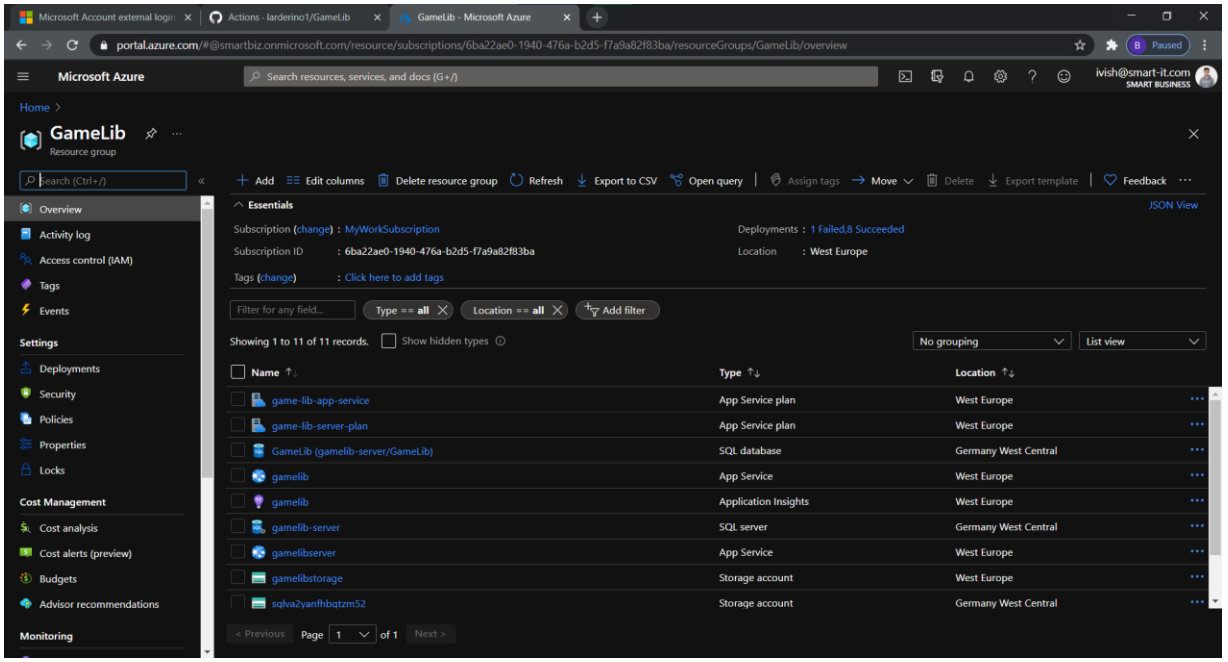
Додаток 6 – сторінка керування категоріями



Додаток 7 – сторінка керування користувачами



Додаток 8 – хмарна інфраструктура Azure



Додаток 9 – контролер керування іграми

```
namespace GameLib_Back.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class GamesController : ControllerBase
    {
        private readonly IGameServices _gamesService;

        public GamesController(IGameServices gamesService)
        {
            _gamesService = gamesService;
        }

        // GET: api/Games
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Game>>> GetGames()
        {
            var games = await _gamesService.GetGameListAsync();

            if(games == null || games.Count() == 0)
            {
                return NoContent();
```

```
    }

    return Ok(games);
}

// GET: api/Games/5
[HttpGet("{id}")]
public async Task<ActionResult<Game>> GetGame(Guid id)
{
    var game = await _gamesService.GetGameByIdAsync(id);

    if (game == null)
    {
        return NotFound();
    }

    return game;
}

// PUT: api/Games/5
[HttpPut("{id}")]
public async Task<IActionResult> PutGame(Guid id, Game game)
{
    if (id != game.Id)
    {
        return BadRequest();
    }

    try
    {
        await _gamesService.UpdateGameAsync(id, game);
    }
    catch (ArgumentNullException ex)
    {
        return NotFound(ex.Message);
    }

    return NoContent();
}

// POST: api/Games
[HttpPost]
public async Task<ActionResult<Game>> PostGame(Game game)
{
    try
    {
        await _gamesService.CreateGameAsync(game);
    }
```

```
            catch(ArgumentNullException ex)
            {
                return BadRequest(ex.Message);
            }

            return CreatedAtAction("GetGame", new { id = game.Id }, game);
        }

        // DELETE: api/Games/5
        [HttpDelete("{id}")]
        public async Task<ActionResult<Game>> DeleteGame(Guid id)
        {
            var game = await _gamesService.DeleteGameAsync(id);

            if (game == null)
            {
                return NotFound();
            }

            return game;
        }
    }
}
```

Додаток 10 – сервіс керування іграми

```
namespace GameLib_Back.Services.GameServices
{
    public class GameServices : IGameServices
    {
        private readonly ApplicationDbContext _context;

        public GameServices(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<Game> CreateGameAsync(Game game)
        {
            try
            {
                await _context.Games.AddAsync(game);
                await _context.SaveChangesAsync();
            }
            catch (ArgumentNullException ex)
            {
                throw new ArgumentNullException("Game can`t be null",
ex.InnerException);
            }
```

```
    return game;
}

public async Task<Game> DeleteGameAsync(Guid id)
{
    var game = await _context.Games.FindAsync(id);

    if (game == null)
    {
        return game;
    }

    _context.Games.Remove(game);
    await _context.SaveChangesAsync();

    return game;
}

public async Task<Game> GetGameByIdAsync(Guid id)
{
    return await _context.Games
        .AsNoTracking()
        .Include(a => a.Category)
        .Include(a => a.Platform)
        .Include(a => a.Mode)
        .Include(a => a.Genre)
        .FirstOrDefaultAsync(a => a.Id == id);
}

public async Task<IEnumerable<Game>> GetGameListAsync()
{
    return await _context.Games
        .AsNoTracking()
        .Include(a => a.Category)
        .Include(a => a.Platform)
        .Include(a => a.Mode)
        .Include(a => a.Genre)
        .ToListAsync();
}

public async Task UpdateGameAsync(Guid id, Game game)
{
    _context.Entry(game).State = EntityState.Modified;

    try
    {
        _context.Games.Update(game);
```

```
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException ex)
            {
                if (!GameExists(id))
                {
                    throw new ArgumentNullException("Game not found", ex);
                }
                else
                    throw ex;
            }
        }

        private bool GameExists(Guid id)
        {
            return _context.Games.Any(e => e.Id == id);
        }
    }
}
```

Календарний план виконання роботи:

№ п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	20.11.2020	
2.	Ознайомлення з предметною областю.	21.11. 2020	
3.	Спілкування з куратором стосовно курсової роботи	21.12.2020	
4.	Пошук літератури та написання першого розділу	25.02.2021	
5.	Побудова структури веб-застосування (другий розділ)	14.03.2021	
6.	Розробка архітектури веб-застосування	10.03.2021	
7.	Початок виконання практичної частини	10.03.2021	
8.	Написання третього розділу	15.04.2021	
9.	Аналіз виконаної роботи з керівником	07.04.2021	
10.	Створення слайдів презентації та написання тексту доповіді	09.04.2021	
11.	Корегування роботи	10.04.2021	
12.	Здача роботи на перевірку на плагіат.	12.04.2021	
13.	Захист курсової роботи	19.04.2021 – 25.04.2021	

Студент **Іщенко І.О.**

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