Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Навчально-науковий інститут природничо-математичних, медико-біологічних наук та інформаційних технологій

Кафедра інформаційних технологій, фізико-математичних та

економічних наук

Освітня програма: Комп’ютерні науки

Спеціальність:122 Компʼютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня *магістр*

**Розробка інформаційної системи обліку поселення в гуртожиток**

студента Лазаковича Євгена Віталійовича

**Науковий керівник:**

Казачков Іван Васильович,

доктор технічних наук, професор

**Рецензенти:**

Фетісов Валерій Сергійович,

Кандидат економічних наук, доцент

Нестеренко Олександр Васильович,

Доктор технічних наук, професор

**Допущено до захисту:** \_\_\_\_\_\_\_\_\_\_ 2024 р.

Завідувач кафедри

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Казачков І.В.

Ніжин − 2024

ЗМІСТ

ВСТУП\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ГУРТОЖИТКА І ЙОГО ПОСЕЛЕННЯ\_\_\_6

1.1 Основні поняття\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

РОЗДІЛ 2. АВТОМАТИЗАЦІЯ ПОСЕЛЕННЯ В ГУРТОЖИТОК\_\_\_\_\_\_\_\_\_\_7

2.1 Важливість автоматизації поселення в гуртожиток\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

2.2 Перелік процесів в гуртожитку, які слід автоматизувати\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

2.3 Програмні рішення для автоматизації процесу поселення в гуртожиток\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

РОЗДІЛ 3. ХАРАКТЕРИСТИКА СУБД MSSQL. КЛІЄНТ MSS MANAGEMENT STUDIO\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

3.1 Загальний опис MSSQL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

3.2 Основні особливості\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

3.3 Основні переваги MSSQL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

3.4 Основні недоліки MSSQL\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12

3.5 Клієнт MSS Management Studio\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

4.1 Встановлення необхідного програмного забезпечення\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

4.1.1 Встановлення Microsoft Visual Studio 2022\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

4.1.2 Встановлення SSMS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_16

4.1.3 Встановлення Postman\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_16

4.2 Створення хмарного середовища для бази даних і сервера\_\_\_\_\_\_\_\_\_\_\_17

4.3 Створення проєкту в Microsoft Visual Studio і підключення до бази даних в хмарному середовищі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19

4.4 Підключеня SSMS до бази даних в хмарному середовищі\_\_\_\_\_\_\_\_\_\_\_\_20

4.5 Створення сутностей і міграція в базу даних\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_21

4.6 Структура бази даних\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_27

4.6.1 Таблиця Hostels\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_27

4.6.2 Таблиця Persons\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_28

4.6.3 Таблиця Rooms\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_29

4.7 Створення Data Transfer Object (DTO)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_30

4.7.1 AutoMapper встановлення і використання\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_32

4.8 Репозиторій і сервіси\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_35

4.8.1 Поняття репозиторію\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_35

4.8.2 Створення репозиторію\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_35

4.8.3 Використання сервісів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_38

4.9 Команди. Патерн Медіатор. Бібліотека MediatR\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_44

4.10 Ендпоінти. Бібліотека Ardalis\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_49

4.11 Middleware\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_51

ВИСНОВКИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_54

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_55

ВСТУП

**Актуальність теми**. Гуртожиток – є невід’ємною частиною життя нетутешнього студента. В ньому людина починає по-справжньому вже доросле життя, де виривається з батьківської хати і вчиться бути самостійною людиною, заводить нові знайомства і вчиться взаємодіяти з новими правилами середовища. Такі правила встановлює саме гуртожиток і його керівники. Для того, щоб правила були правильними і не утискали мешканців, важливо, щоб поселення було структурованим і розподіленим. Раніше це були нескінченні рукописні журнали обліку, де щоб знайти інформацію про мешканця треба було дуже багато часу листати їх. А що робити, якщо інформація змінюється? Черкати рукописні записи і наводити нові? Заздалегідь писати олівцем а потім стирати ластиком? В часи цифровізації це зовсім недоцільно. Тим паче це додаткова популяризація використання комп’ютерів серед працівників гуртожитків і навчання їх новим технологіям.

Отже, з огляду на значущість ведення обліку мешканців гуртожитку, вивчення та впровадження сучасних підходів до автоматизації її процесів має великий практичний і науковий інтерес.

**Об’єкт дослідження** – серверна частина системи обліку поселення.

**Предмет дослідження** – використання системи обліку для автоматизації процесів поселення мешканців в гуртожиток.

**Метою** кваліфікаційної роботи є розробка серверної частини системи обліку поселення в гуртожиток.

Відповідно до вказаного об’єкту, предмету і для досягнення поставленої мети визначені такі **завдання** роботи:

* проаналізувати технічну, спеціальну літературу відповідно до теми кваліфікаційної роботи;
* знайти різноманітні ефективні способи реалізації потрібної задачі з використанням баз даних, бібліотек, вивчити можливості знайдених доступних прикладних рішень та особливості роботи з такими .NET Core 8.0;
* реалізувати задачу API викликів з UI частини, обробка вищесказаних викликів і віддача відповідей;
* розробка автоматизації і надання ролей для різноманітних працівників гуртожитку, надання безпекових рішень задля запобіганню втручання ззовні;
* зробити висновки щодо ефективності та переваг цифрового ведення обліку поселення в гуртожиток.

Робота має **практичне значення**, оскільки результати дослідження можуть бути використані надалі гуртожитками різних структур задля ведення і зберігання даних в цифорвому форматі, що безпосередньо може принести вигоду обом сторонам.

РОЗДІЛ 1

ХАРАКТЕРИСТИКА ГУРТОЖИТКА І ЙОГО ПОСЕЛЕННЯ

1.1 Основні поняття

Гуртожиток - це спеціально споруджений або переобладнаний жилий будинок, який використовується для проживання робітників, службовців, студентів, учнів, а також інших громадян у період роботи або навчання. Жилі будинки реєструються як гуртожитки у виконавчому комітеті відповідної ради. За призначенням гуртожитки поділяються на два види:

* для проживання окремих осіб (жилі приміщення перебувають у користуванні кількох осіб, які не перебувають між собою в сімейних відносинах);
* для проживання сімей (жилі приміщення, що складаються з однієї чи кількох кімнат, перебувають у відособленому користуванні однієї сім’ї).

Не допускається розміщення гуртожитків у підвалах і цокольних поверхах, а також використання під гуртожитки приміщень у жилих будинках, призначених для постійного проживання громадян.

Жила площа в гуртожитку не підлягає обміну, розділу, бронюванню і здачі в піднайом.

РОЗДІЛ 2

АВТОМАТИЗАЦІЯ ПОСЕЛЕННЯ В ГУРТОЖИТОК

2.1 Важливість автоматизації поселення в гуртожиток

Автоматизація процесу поселення в гуртожиток за допомогою програмного забезпечення відкриває нові горизонти для оптимізації адміністративних задач.

* По-перше, це значно зменшує час на оформлення поселення, оскільки всі документи можуть бути подані та оброблені онлайн. Працівники можуть швидко заповнити необхідні форми, завантажити документи та отримати підтвердження в режимі реального часу. Це скорочує черги та знижує навантаження на адміністративний персонал.
* По-друге, автоматизація знижує ризик помилок при обробці даних. Система може автоматично перевіряти наявність необхідних документів, відповідність даних та їх правильність. Це підвищує точність та надійність всього процесу. Адміністрація має можливість легко контролювати та управляти даними, що сприяє кращому обліку студентів і більш ефективному використанню ресурсів.
* По-третє, програмне забезпечення дозволяє студентам самостійно обирати та бронювати кімнати. Вони можуть бачити доступні варіанти, отримувати інформацію про умови проживання та робити свій вибір на основі особистих уподобань. Це підвищує рівень задоволеності студентів, адже вони можуть самостійно керувати своїм проживанням.

Загалом, автоматизація процесу поселення робить його більш прозорим, зручним та ефективним, що сприяє покращенню досвіду як для студентів, так і для адміністративного персоналу.

2.2 Перелік процесів в гуртожитку, які слід автоматизувати

Автоматизація процесів у гуртожитку значно підвищує ефективність управління та комфорт проживання студентів. Ось кілька ключових процесів, які можна автоматизувати:

1. Поселення та виселення: Автоматизовані системи можуть допомагати у подачі заявок, реєстрації нових мешканців та процесу виселення. Це включає в себе електронне підписання договорів, підтвердження бронювання кімнат та перевірку документів.

2. Управління кімнатами: Студенти в майбутньому можуть самостійно обирати та бронювати кімнати через онлайн-платформи, що дозволяє уникнути помилок при розподілі кімнат та підвищує задоволеність мешканців.

3. Облік платежів: В майбутньому система може автоматично обробляти платежі за проживання, інформувати студентів про необхідність оплати та надсилати нагадування. Це полегшує фінансовий облік для адміністрації та зменшує кількість прострочених платежів.

4. Звіти про технічні проблеми: Студенти можуть онлайн повідомляти про технічні несправності або інші проблеми в гуртожитку. Система може автоматично розподіляти заявки на відповідні служби та контролювати виконання робіт.

5. Інформаційні розсилки: Автоматизація дозволяє легко інформувати мешканців про важливі новини та події, відправляючи електронні листи або повідомлення через додаток.

6. Безпека: Системи автоматизації можуть включати електронні пропуски та відеоспостереження для підвищення рівня безпеки в гуртожитку.

Загалом, автоматизація покращує організацію роботи гуртожитку, підвищуючи ефективність, точність та задоволеність мешканців.

2.3 Програмні рішення для автоматизації процесу поселення в гуртожиток

В сучасному світі паперові журнали відходять на другий план і їх заміняють автоматизовані програми. Але які механізми краще використовувати задля розробки процесу поселення в гуртожиток?

Використання .NET Core 8.0 для розробки серверної частини забезпечує високу продуктивність і масштабованість системи, що дозволяє ефективно обробляти великі обсяги даних. Це важливо для швидкої обробки заявок на поселення, перевірки документів та розподілу кімнат.

База даних MSSQL, використовувана в системі, гарантує надійне зберігання та швидкий доступ до даних. Це дозволяє зберігати всю інформацію про студентів, кімнати, платежі та технічні заявки в одному місці, забезпечуючи легкий доступ та керування.

Azure, як хмарна платформа, забезпечує гнучкість та надійність інфраструктури. Це дозволяє системі працювати без перебоїв, забезпечуючи високу доступність сервісу навіть при пікових навантаженнях, таких як початок навчального року або період виселення. А також мультизадачність, де ми можемо зберігати як базу даних так і програмне рішення в одному місці.

РОЗДІЛ 3

ХАРАКТЕРИСТИКА СУБД MSSQL. КЛІЄНТ MSS MANAGEMENT STUDIO

3.1 Загальний опис MSSQL

MSSQL - система управління базами даних, яка розробляється корпорацією Microsoft. Як сервер даних виконує головну функцію по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватися як на тому ж самому сервері, так і у мережі.

Мова, що використовується для запитів — Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.

3.2 Основні особливості

MSSQL є потужною системою управління реляційними базами даних (RDBMS), що має кілька ключових особливостей, які роблять її популярною серед розробників та адміністраторів баз даних.

1. Висока продуктивність: MSSQL забезпечує швидке виконання запитів і обробку даних завдяки оптимізованому двигуну бази даних. Інструменти, такі як індексація та кешування, допомагають підвищити ефективність.

2. Безпека: MSSQL має розширені можливості захисту даних, включаючи шифрування, контроль доступу на основі ролей (RBAC) та аудит. Це дозволяє забезпечити високу безпеку даних і відповідність стандартам.

3. Масштабованість: Платформа підтримує як локальні, так і хмарні розгортання, що дозволяє легко адаптуватися до зростання обсягів даних і збільшення кількості користувачів.

4. Інтеграція з іншими продуктами Microsoft: MSSQL легко інтегрується з такими інструментами, як Power BI, Azure, Microsoft Office та інші, що розширює його функціональні можливості і полегшує аналіз даних.

5. Розширені можливості для аналітики та звітності: Завдяки вбудованим функціям для бізнес-аналітики (BI) та потужним засобам звітності, MSSQL дозволяє проводити глибокий аналіз даних і створювати складні звіти.

6. Підтримка різних мов програмування: MSSQL підтримує T-SQL для написання запитів, а також інтегрується з мовами, такими як C#, Python та R, для розробки та аналітики.

Ці особливості роблять MSSQL універсальним інструментом для управління базами даних, що задовольняє потреби як малих підприємств, так і великих корпорацій.

3.3 Основні переваги MSSQL

Microsoft SQL Server (MSSQL) має багато переваг, які роблять його популярним вибором для управління базами даних:

1. Висока продуктивність: MSSQL оптимізований для швидкого виконання запитів і ефективної обробки великих обсягів даних. Це забезпечується завдяки індексації, кешуванню та іншим технологіям, що покращують продуктивність.

2. Безпека: MSSQL забезпечує потужні засоби захисту даних, включаючи шифрування, контроль доступу на основі ролей (RBAC) та аудит. Це дозволяє забезпечити конфіденційність та цілісність даних.

3. Масштабованість: MSSQL підтримує горизонтальне та вертикальне масштабування, що дозволяє легко адаптувати систему до зростаючих потреб організації.

4. Інтеграція з іншими продуктами Microsoft: MSSQL легко інтегрується з такими продуктами, як Power BI, Azure та Microsoft Office, що забезпечує широкі можливості для аналізу та обробки даних.

5. Підтримка високої доступності та відновлення після збоїв: MSSQL має вбудовані функції для забезпечення безперебійної роботи системи, такі як кластеризація, реплікація та механізми відновлення після збоїв.

6. Аналітика та звітність: MSSQL включає потужні засоби для бізнес-аналітики та створення звітів, дозволяючи проводити глибокий аналіз даних та генерувати деталізовані звіти.

7. Підтримка різних мов програмування: MSSQL підтримує T-SQL для написання запитів, а також інтегрується з мовами, такими як C#, Python, R, що забезпечує широкі можливості для розробки та аналізу.

Завдяки цим перевагам, MSSQL є універсальним та надійним рішенням для організацій будь-якого масштабу, забезпечуючи ефективне управління даними.

3.4 Основні недоліки MSSQL

Microsoft SQL Server (MSSQL) має багато переваг, але, як і будь-яка система, має також деякі недоліки:

1. Вартість: MSSQL є комерційним продуктом, що вимагає оплати за ліцензію, особливо у випадку великих корпорацій або при використанні розширених функцій. Це може бути вагомим фактором для невеликих компаній або стартапів.

2. Ресурсомісткість: MSSQL може вимагати значних апаратних ресурсів, особливо для великих баз даних або при високих навантаженнях. Це може збільшити витрати на сервери та обслуговування.

3. Складність налаштування: Попри те, що MSSQL має багато інструментів для адміністрування, налаштування системи може бути складним і вимагати висококваліфікованих спеціалістів. Це може створювати труднощі для компаній, які не мають великої ІТ-команди.

4. Закритий код: На відміну від деяких інших систем управління базами даних, MSSQL має закритий вихідний код. Це може обмежити можливості для кастомізації або інтеграції з іншими системами.

5. Складність масштабування: Хоча MSSQL підтримує масштабування, цей процес може бути складним та вимагати додаткових ресурсів та інструментів для досягнення оптимальної продуктивності.

6. Залежність від Windows: Хоча останні версії MSSQL підтримують Linux, історично система була тісно пов'язана з екосистемою Windows. Це може створювати проблеми для організацій, що використовують інші операційні системи.

Ці недоліки можуть впливати на вибір MSSQL як основної системи управління базами даних, але важливо зважити всі аспекти перед прийняттям рішення.

3.5 Клієнт MSS Management Studio

Microsoft SQL Server Management Studio (SSMS) є інтегрованою средою для управління будь-якою SQL-інфраструктурою, включаючи SQL Server, Azure SQL Database та інші. SSMS надає інструменти для налаштування, моніторингу та адміністрування екземплярів SQL Server та баз даних. Використовується для розгортання, моніторингу та оновлення компонентів даних, які використовуються вашими додатками.

SSMS дозволяє запитувати, проектувати та управляти базами даних та даними, будь-які вони знаходяться на вашому локальному комп'ютері або в хмарі. SSMS також включає широкий набір графічних інструментів та редакторів скриптів, що дозволяє розробникам та адміністраторам баз даних різних рівнів кваліфікації працювати з SQL Server.

Основні компоненти SSMS включають Object Explorer для перегляду та управління всіма об'єктами в одному або більше екземплярів SQL Server, Template Explorer для створення та управління шаблонами тексту, а також Visual Database Tools для створення запитів, таблиць та діаграм баз даних.

SSMS також підтримує адміністрування служб аналізу, інтеграції та звітування, хоча для створення та модифікації рішень для цих служб використовується SQL Server Data Tools (SSDT). SSMS надає можливість створювати та зберігати скрипти на основі Multidimensional Expressions (MDX), Data Analysis Expressions (DAX), Data Mining Extensions (DMX) та XML for Analysis (XMLA).

Загалом, SSMS є незамінним інструментом для управління SQL Server, що забезпечує всі необхідні функції для розробників та адміністраторів баз даних.

РОЗДІЛ 4

ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1 Встановлення необхідного програмного забезпечення

Для початку реалізації серверної частини проєкту системи обліку поселення в гуртожиток необхідно встановити наступні програми:

* Microsoft Visual Studio 2022
* Microsoft SSM Studio 19
* Postman

4.1.1 Встановлення Microsoft Visual Studio 2022

Для початку перейдемо на офіційний сайт Microsoft Visual Studio <https://visualstudio.microsoft.com/>

Натискаємо Download і завантажуємо інсталяційний файл собі на комп’ютер.

Встановлюємо Installer. А потім обираємо стандартну установку.

A screenshot of a computer

Description automatically generated

Рисунок 1 – Visual Studio Installer

4.1.2 Встановлення SSMS

Заходимо на офіційну сторінку SSMS <https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16> і натискаємо на Download SSMS.

Встановлюємо Installer. Обираємо стандартну установку.

A screenshot of a computer

Description automatically generated

Рисунок 2 – SSMS Installer

4.1.3 Встановлення Postman

Заходимо на офіційний сайт Postman <https://www.postman.com/>

Натискаємо Download. Інсталюємо програму.

A screenshot of a computer

Description automatically generated

Рисунок 3 – Інтерфейс програми Postman

4.2 Створення хмарного середовища для бази даних і сервера

Для повноційної роботи обліку поселення студентів в гуртожиток нам необхідно вивести в «хмару» наші елементи, такі як база даних та сервер. Для цього нам знадобиться хмарне середовище. Будемо використовувати Azure.

Azure - хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень (англ. *cloud computing*) і покликана спростити процес створення онлайнових додатків.

Для зручного і структурованого зберігання всіх наших елементів будемо використовувати ресурсну групу.

A screenshot of a computer

Description automatically generated

Рисунок 4 – Інтерфейс списка ресурних груп

Натискаємо Create.

A screenshot of a computer

Description automatically generated

Рисунок 5 – Інтерфейс створення ресурсної групи

Обираємо потрібну нам підписку і прописуємо ім’я нашої групи. Після цього натискаємо Створити і ресурсна група в нас створена. Далі створюємо app service, sql server, sql database

A screenshot of a computer

Description automatically generated

Рисунок 6 – Інтерфейс елементів rg-hostel-api ресурсної групи

App service буде зберігати нашу бекенд програму, в sql database буде зберігатись наша база даних, sql server буде нашим рушієм бази даних.

4.3 Створення проєкту в Microsoft Visual Studio і підключення до бази даних в хмарному середовищі

Запускаємо Microsoft Visual Studio і обираємо Create а new project

A screenshot of a computer

Description automatically generated

Рисунок 7 – Інтерфейс створення проєкта в Microsoft Visual Studio

Обираємо ASP.NET Core Empty версія 8.0, обираємо назву і натискаємо кнопку Create

A screenshot of a computer program

Description automatically generated

Рисунок 8 – Інтерфейс створення проєкта в Microsoft Visual Studio

В файлі appsettings.json додаємо нашу строку підключення до бази даних в форматі:

"DbConnection": "Server=tcp:hostel-api-db-server.database.windows.net,1433;Initial Catalog=hostel-api-db;Persist Security Info=False;User ID=<YOUR\_LOGIN>;Password=<YOUR\_PASSWORD>; MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Authentication=\"Active Directory Password\";"

4.4 Підключеня SSMS до бази даних в хмарному середовищі

Відкриваємо SSMS і вводимо в Server name Server=tcp:hostel-api-db-server.database.windows.net,1433;, використовуємо наш логін і пароль від Azure і підставляємо їх в значення Login, Password. Спосіб аутентифікації обираємо Microsoft Entra Password

A screenshot of a computer

Description automatically generated

Рисунок 9 – Підключення до бази даних в SSMS

4.5 Створення сутностей і міграція в базу даних

Для з’єднання наших сутностей в програмі і таблиць в базі даних будемо використовувати Microsoft Entity Framework Core.

A screenshot of a computer

Description automatically generated

Рисунок 10 – Встановлення Microsoft Entity Framework Core

Ця бібліотека дозволяє мігрувати наші сутності в вигляді класів в базу даних і щоб вони співпадали між собою, щоб не було випадків, що в програмі значення числове, а в коді, наприклад, строкове.

Додаємо клас ApplicationDbContext.cs

A computer screen shot of a program

Description automatically generated

Рисунок 11 – Клас ApplicationDbContext.cs

Цей клас відповідає за зв’язок з базою даних і перелік потрібних таблиць в базі

Додаємо нашу строку підключення в Program.cs:

*builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DbConnection"), b => b.MigrationsAssembly("Hostel.Core")), ServiceLifetime.Scoped);*

Це означає, що ми підключаємо саме цю строку і використовуємо її як строку під’єднання до бази даних.

Додаємо наші сутності – Hostel, Room, Person, а також загальну сутність BaseEntity, яка буде мати загальні властивості для всіх сутностей, такі як Id, CreatedDate, ModifiedDate, IsDelete:

*public class BaseEntity*

*{*

*public BaseEntity()*

*{*

*Id = Guid.NewGuid();*

*CreatedDate = DateTime.UtcNow;*

*}*

*[Required]*

*public Guid Id { get; set; }*

*[Required]*

*public DateTime CreatedDate { get; set; }*

*public DateTime? ModifiedDate { get; set; }*

*public bool IsDelete { get; set; } = false;*

*}*

*public class HostelEntity : BaseEntity*

*{*

*public string Number { get; set; }*

*public string Address { get; set; }*

*}*

*public class PersonEntity : BaseEntity*

*{*

*public string LastName { get; set; }*

*public string FirstName { get; set; }*

*public string FatherName { get; set; }*

*public string Faculty { get; set; }*

*public string PhoneNumber { get; set; }*

*public string PayType { get; set; }*

*public int PaySum { get; set; }*

*public string AdditionalInformation { get; set; }*

*public Guid? RoomId { get; set; }*

*public RoomEntity Room { get; set; }*

*}*

*public class RoomEntity : BaseEntity*

*{*

*public Guid HostelId { get; set; }*

*public HostelEntity Hostel { get; set; }*

*public int Level { get; set; }*

*public int Number { get; set; }*

*public int Size { get; set; }*

*public string AdditionalInformation { get; set; }*

*public string StuffInformation { get; set; }*

*public List<PersonEntity> Persons { get; set; }*

*}*

Тепер після створення сутностей нам потрібно мігрувати їх до бази даних. Будемо використовувати NuGet Package Manager Console

A screenshot of a computer program

Description automatically generated

Рисунок 12 – Додавання міграції

Додаємо команду Add-Migration Initial, яка створить міграцію з назвою Initial

A screen shot of a computer program

Description automatically generated

Рисунок 13 – Міграція Initial

Entity Framework Core створила автоматично клас Initial, в якій можна побачити які зміни відбудуться в базі даних. Є два методи Up i Down. Up слугує для того, щоб внести зміни при оновлені бази даних, Down, в свою чергу, слугує для того, щоб відкатити зміни, якщо на це буде потреба.

Тепер треба оновити базу даних. Для цього знову відкриваємо NuGet Package Manager Console і вводимо команду Update-Database

A screenshot of a computer program

Description automatically generated

Рисунок 14 – Оновлення бази даних

Після виклику команди ми можемо помітити, що в базу додались таблиці, які нас цікавлять, а саме: Hostels, Persons, Rooms, а також стандартні таблиці для авторизації і аутентифікації

A screenshot of a computer

Description automatically generated

Рисунок 15 – Список таблиць в SSMS

Таким чином ми можемо вносити зміни в базу даних, якщо для цього є необхідність і пов’язувати безпосередні наші сутності з таблицею в базі даних. Такий метод називається code-first, коли спочатку створюються сутності, а потім вони переносяться в базу даних таблицею.

4.6 Структура бази даних

Як було зазначено вище був використаний метод code-first, де було створено сутності, а потім вони мігрували в базу даних. Але що ж тепер в базі даних знаходиться?

Підключаємось до бази даних за допомогою SSMS, знаходимо наші таблиці, на кожну клікаємо правою кнопкою миші і обираєм варіант Design.

Переглянемо таким чином таблиці Hostels, Persons, Rooms.

4.6.1 Таблиця Hostels

A screenshot of a computer

Description automatically generated

Рисунок 16 – Структура таблиці Hostels

Таблиця має 6 полів (Id, Number, Address, CreatedDate, ModifiedDate, IsDelete). Поля Id, CreatedDate, ModifiedDate, IsDelete є спільними для всіх таблиць, що є похідними властивостями від сутності BaseEntity.

* Id – ідентифікатор запису, є Primary Key, формату Guid
* Number – номер гуртожитка, nvarchar
* Address – адреса гуртожитку, nvarchar
* CreatedDate – дата створення запису, datetime
* ModifiedDate – дата редагування запису, datetime, nullable
* IsDelete – ознака видаленого запису, bool

В цьому проєкті використовується правило м’якого видалення (soft delete), де ми не видаляємо запис повністю, а лише змінюємо ознаку IsDelete = true.

4.6.2 Таблиця Persons

A screenshot of a data

Description automatically generated

Рисунок 17 – Структура таблиці Persons

* Id – ідентифікатор запису, є Primary Key, формату Guid
* LastName – прізвище мешканця, nvarchar
* FirstName – ім’я мешканця, nvarchar
* FatherName – по-батькові мешканця, nvarchar
* PayType – спосіб оплати, nvarchar
* PhoneNumber – номер телефона, nvarchar
* AdditionalInformation – додаткова інформація або коментар, nvarchar
* CreatedDate – дата створення запису, datetime
* ModifiedDate – дата редагування запису, datetime, nullable
* IsDelete – ознака видаленого запису, bool
* RoomId – ідентифікатор кімнати, Guid, nullable
* Faculty – назва факультету, nvarchar
* PaySum – розмір оплати, integer

4.6.3 Таблиця Rooms

A screenshot of a computer

Description automatically generated

Рисунок 18 – Структура таблиці Rooms

* Id – ідентифікатор запису, є Primary Key, формату Guid
* HostelId – ідентифікатор гуртожитка, Guid
* Level – номер поверха, integer
* Number – номер кімнати, integer
* Size – розмір кімнати, integer
* CreatedDate – дата створення запису, datetime
* ModifiedDate – дата редагування запису, datetime, nullable
* IsDelete – ознака видаленого запису, bool
* AdditionalInformation – додаткова інформація або коментар, nvarchar
* StuffInformation – інформація про майно, nvarchar

Цих таблиць вистачить для нашого проєкту, що забезпечить коректну роботу програми по обліку поселення в гуртожиток

4.7 Створення Data Transfer Object (DTO)

Задля комфортного і оптимізованого спілкування фронт і бек частин програми існують DTO. По суті DTO – це ті класи, які використовуються при тому чи іншому запиті. Так само як і сутності, буде загальна DTO (BaseItemDto), від якої похідні будуть наслідковуватись.

В цьому проєкті нам знадобляться такі DTO:

*public class BaseItemDto*

*{*

*public Guid? Id { get; set; }*

*public DateTime? CreatedDate { get; set; }*

*public DateTime? ModifiedDate { get; set; }*

*public bool IsDelete { get; set; }*

*}*

*public class CreatePersonDto : BaseItemDto*

*{*

*public PersonDto Person { get; set; }*

*public Guid? RoomId { get; set; }*

*}*

*public class CreateRoomItemDto : BaseItemDto*

*{*

*public Guid HostelId { get; set; }*

*public int Level { get; set; }*

*public int Number { get; set; }*

*public int Size { get; set; }*

*}*

*public class HostelDto : BaseItemDto*

*{*

*public string Number { get; set; }*

*public string Address { get; set; }*

*}*

*public class PersonDto : BaseItemDto*

*{*

*public string LastName { get; set; }*

*public string FirstName { get; set; }*

*public string FatherName { get; set; }*

*public string Faculty { get; set; }*

*public string PhoneNumber { get; set; }*

*public string PayType { get; set; }*

*public int PaySum { get; set; }*

*public string AdditionalInformation { get; set; }*

*}*

*public class RoomItemDto : BaseItemDto*

*{*

*public string AdditionalInformation { get; set; }*

*public string StuffInformation { get; set; }*

*public int Number { get; set; }*

*public int Size { get; set; }*

*public List<PersonDto> Persons { get; set; } = new();*

*}*

*public class RoomListItemDto : BaseItemDto*

*{*

*public Guid HostelId { get; set; }*

*public string HostelNumber { get; set; }*

*public int Level { get; set; }*

*public int Number { get; set; }*

*public int Size { get; set; }*

*public int PersonCount { get; set; }*

*}*

*public class RoomStatisticDto : BaseItemDto*

*{*

*public int PersonCountByLevel { get; set; }*

*public int PersonCountByHostel { get; set; }*

*public int PersonSumByLevel { get; set; }*

*public int PersonSumByHostel { get; set; }*

*public double PersonPercentageLevelByLevel { get; set; }*

*public double PersonPercentageLevelByHostel { get; set; }*

*}*

*public class SavePersonDto*

*{*

*public Guid PersonId { get; set; }*

*public Guid? RoomId { get; set; }*

*}*

Як було зазначено вище DTO – це класи, які ми використовуємо для прийому запиту і віддачі відповіді. А як конвертувати їх в сутності? Для цього нам допоможе бібліотека AutoMapper

4.7.1 AutoMapper встановлення і використання

AutoMapper дозволяє проектувати одну модель на іншу, що дозволяє скоротити обсяги коду та спростити програму.

Встановимо AutoMapper за допомогою NuGet

A screenshot of a computer

Description automatically generated

Рисунок 19 – Встановлення AutoMapper

Після встановлення створюємо клас BaseMappingProfile.cs і додаємо наші правила перетворення з одного класа в інший, це значно скоротить час і обсяг роботи

*public class BaseMappingProfile : Profile*

*{*

*public BaseMappingProfile()*

*{*

*CreateMap<BaseEntity, BaseItemDto>()*

*.ForMember(dest => dest.Id, opt => opt.MapFrom(x => x.Id));*

*CreateMap<BaseItemDto, BaseEntity>()*

*.ForMember(dest => dest.Id, opt => opt.MapFrom(x => x.Id));*

*CreateMap<HostelEntity, HostelDto>().IncludeBase<BaseEntity, BaseItemDto>();*

*CreateMap<HostelDto, HostelEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*CreateMap<RoomEntity, RoomListItemDto>().IncludeBase<BaseEntity, BaseItemDto>()*

*.ForMember(dest => dest.HostelNumber, opt => opt.MapFrom(x => x.Hostel != null ? x.Hostel.Number : string.Empty))*

*.ForMember(dest => dest.PersonCount, opt => opt.MapFrom(x => x.Persons != null ? x.Persons.Count : 0));*

*CreateMap<RoomListItemDto, RoomEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*CreateMap<RoomEntity, RoomItemDto>().IncludeBase<BaseEntity, BaseItemDto>();*

*CreateMap<RoomItemDto, RoomEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*CreateMap<RoomEntity, CreateRoomItemDto>().IncludeBase<BaseEntity, BaseItemDto>();*

*CreateMap<CreateRoomItemDto, RoomEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*CreateMap<PersonEntity, PersonDto>().IncludeBase<BaseEntity, BaseItemDto>();*

*CreateMap<PersonDto, PersonEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*CreateMap<PersonEntity, CreatePersonDto>().IncludeBase<BaseEntity, BaseItemDto>();*

*CreateMap<CreatePersonDto, PersonEntity>().IncludeBase<BaseItemDto, BaseEntity>();*

*}*

*}*

4.8 Репозиторій і сервіси

4.8.1 Поняття репозиторію

У .NET Core репозиторій є патерном проєктування, який використовується для абстрагування доступу до даних. Він дозволяє розробникам працювати з різними джерелами даних (наприклад, базами даних, веб-службами) через єдиний інтерфейс. Репозиторій забезпечує централізоване місце для керування операціями з даними, такими як створення, читання, оновлення та видалення (CRUD-операції).

Основна перевага патерну репозиторію полягає в тому, що він сприяє розподілу відповідальностей та спрощує тестування. Клієнтський код взаємодіє з даними через інтерфейси, що полегшує заміну реальних реалізацій на фейкові (mock) при написанні юніт-тестів. Це підвищує гнучкість і підтримуваність коду.

4.8.2 Створення репозиторію

Створимо клас DbRepositoryBase.cs.

*public class DbRepositoryBase*

*{*

*private readonly IUnitOfWorkBase unitOfWork;*

*public DbRepositoryBase(IUnitOfWorkBase unitOfWork)*

*{*

*this.unitOfWork = unitOfWork;*

*}*

*public IQueryable<T> FindAll<T>(Expression<Func<T, bool>> where = null, bool deleteAvailable = false) where T : BaseEntity*

*{*

*return null != where ? unitOfWork.Set<T>().Where(c => deleteAvailable || !c.IsDelete).Where(where).OrderByDescending(c => c.CreatedDate) : unitOfWork.Set<T>().Where(c => !c.IsDelete).OrderByDescending(c => c.CreatedDate);*

*}*

*public T Find<T>(Expression<Func<T, bool>> where = null) where T : BaseEntity*

*{*

*return FindAll(where, true).FirstOrDefault();*

*}*

*public bool Any<T>(Expression<Func<T, bool>> where = null) where T : BaseEntity*

*{*

*return unitOfWork.Set<T>().Any(where);*

*}*

*public T Create<T>(T entity) where T : BaseEntity*

*{*

*entity.CreatedDate = DateTime.UtcNow;*

*return unitOfWork.Set<T>().Add(entity).Entity;*

*}*

*public void CreateRange<T>(IEnumerable<T> entity) where T : BaseEntity*

*{*

*unitOfWork.Set<T>().AddRange(entity);*

*}*

*public void Update<T>(T entity) where T : BaseEntity*

*{*

*unitOfWork.Entry(entity).State = Microsoft.EntityFrameworkCore.EntityState.Modified;*

*entity.ModifiedDate = DateTime.UtcNow;*

*}*

*public void Delete<T>(int id) where T : BaseEntity*

*{*

*var item = unitOfWork.Set<T>().Find(id);*

*Delete(item);*

*}*

*public void Delete<T>(Guid id) where T : BaseEntity*

*{*

*var item = unitOfWork.Set<T>().Find(id);*

*Delete(item);*

*}*

*public void Delete<T>(T entity) where T : BaseEntity*

*{*

*unitOfWork.Entry(entity).State = Microsoft.EntityFrameworkCore.EntityState.Modified;*

*entity.IsDelete = true;*

*}*

*public int Count<T>(Expression<Func<T, bool>> where = null) where T : BaseEntity*

*{*

*return where != null ?*

*unitOfWork.Set<T>().Where(where).Count() :*

*default;*

*}*

*public async Task SaveChangesAsync()*

*{*

*await unitOfWork.SaveChangesAsync();*

*}*

*}*

Цей клас реалізовує базові команди, які через обгортку Entity Framework Core конвертуються в SQL запити і надсилаються в базу даних, тим самим з’єднуємо наш серверний код безпосередньо в базу даних. Це гнучкий метод спілкування з базою даних, що суттєво скорочує і код і час роботи над цим проєктом.

4.8.3 Використання сервісів

Сервіси слугують для конвертування загального репозиторію в конкретні запити під кожну сутність, тобто сервіс формує під яку саму таблицю буде формуватись запит в базу даних. Наприклад це може бути запит в таблицю Гуртожитків (Hostels) або Кімнат (Rooms) і так далі.

Так як сутності мають якісь схожі елементи (BaseEntity), то можемо використати також загальний сервіс під них, який, ще і до цього, зробимо «дженеріком»:

*public class GenericService<T> where T : BaseEntity, new()*

*{*

*private readonly IDbRepository \_dbRepository;*

*public GenericService(IDbRepository dbRepository)*

*{*

*\_dbRepository = dbRepository;*

*}*

*public virtual async Task<T> GetById(Guid id)*

*{*

*return await \_dbRepository.FindAll<T>(c => c.Id == id, true).FirstOrDefaultAsync();*

*}*

*public virtual async Task<List<T>> GetAll()*

*{*

*return await \_dbRepository.FindAll<T>().ToListAsync();*

*}*

*public virtual async Task Add(T entity)*

*{*

*\_dbRepository.Create(entity);*

*}*

*public virtual async Task Update(T entity)*

*{*

*\_dbRepository.Update(entity);*

*}*

*public virtual async Task Delete(Guid id)*

*{*

*var entity = await GetById(id);*

*if (entity != null)*

*{*

*\_dbRepository.Delete(entity);*

*}*

*}*

*public virtual async Task Delete(T entity)*

*{*

*if (entity != null)*

*{*

*\_dbRepository.Delete(entity);*

*}*

*}*

*public virtual async Task<List<T>> GetByPredicate(Expression<Func<T, bool>> predicate)*

*{*

*return await \_dbRepository.FindAll(predicate).ToListAsync();*

*}*

*public async Task SaveChangesAsync()*

*{*

*await \_dbRepository.SaveChangesAsync();*

*}*

*}*

Цей сервіс викликає наш репозиторій і формує відповіді безпосередньо вже під наші сутності, що дозволяє отримувати коректні і очікувані результати після запиту до бази даних.

Також не слід забувати про конкретні сервіси, які будуть наслідковуватись від GenericServvice.cs і передавати безпосередньо конкретну сутність, щоб зв’язатись з базою даних.

Таких сервісів три – HostelService.cs, PersonService.cs, RoomService.cs

*public class HostelService : GenericService<HostelEntity>*

*{*

*private readonly IDbRepository \_dbRepository;*

*public HostelService(IDbRepository dbRepository) : base(dbRepository)*

*{*

*}*

*}*

*public class PersonService : GenericService<PersonEntity>*

*{*

*private readonly IDbRepository \_dbRepository;*

*public PersonService(IDbRepository dbRepository) : base(dbRepository)*

*{*

*\_dbRepository = dbRepository;*

*}*

*public override async Task<PersonEntity> GetById(Guid id)*

*{*

*var personEntity = await \_dbRepository*

*.FindAll<PersonEntity>(c => c.Id == id)*

*.Include(c => c.Room)*

*.FirstOrDefaultAsync();*

*return personEntity;*

*}*

*public async Task<int> GetCountByPredicate(Expression<Func<PersonEntity, bool>> predicate)*

*{*

*return await \_dbRepository*

*.FindAll(predicate)*

*.Include(c => c.Room)*

*.CountAsync();*

*}*

*}*

*public class RoomService : GenericService<RoomEntity>*

*{*

*private readonly IDbRepository \_dbRepository;*

*public RoomService(IDbRepository dbRepository) : base(dbRepository)*

*{*

*\_dbRepository = dbRepository;*

*}*

*public override async Task<List<RoomEntity>> GetByPredicate(Expression<Func<RoomEntity, bool>> predicate)*

*{*

*return await \_dbRepository*

*.FindAll(predicate)*

*.Include(c => c.Persons)*

*.ToListAsync();*

*}*

*public override async Task<List<RoomEntity>> GetAll()*

*{*

*var roomEntities = await \_dbRepository*

*.FindAll<RoomEntity>()*

*.Include(c => c.Hostel)*

*.ToListAsync();*

*return roomEntities;*

*}*

*public override async Task<RoomEntity> GetById(Guid id)*

*{*

*var roomEntity = await \_dbRepository*

*.FindAll<RoomEntity>(c => c.Id == id)*

*.Include(c => c.Hostel)*

*.FirstOrDefaultAsync();*

*return roomEntity;*

*}*

*public async Task<int> GetCountByPredicate(Expression<Func<RoomEntity, bool>> predicate)*

*{*

*return await \_dbRepository*

*.FindAll(predicate)*

*.CountAsync();*

*}*

*}*

Як можна помітити на цьому етапі вже кожен сервіс має свою реалізацію, що дозволяє масштабуватись під різні потреби програми.

Таким чином ми зв’язуємось з репозиторієм і отримуємо результати, які нам необхідні.

4.9 Команди. Патерн Медіатор. Бібліотека MediatR

Для реалізації бізнес логіки необхідно створити класи, які будуть об’єднувати виклики різних окремих сервісів для загальної мети – опрацювання запиту і віддача результату. В даному проєкті вирішено використовувати патерн Медіатор.

Патерн Посередник (Mediator) представляє такий шаблон проектування, який забезпечує взаємодію багатьох об'єктів без необхідності посилатися один на одного. Тим самим досягається слабозв'язаність взаємодіючих об'єктів.

Інакше кажучи: посередник — це клей, який пов'язує кілька незалежних класів між собою. Він позбавляє класи необхідності посилатися друг на друга, дозволяючи цим їх незалежно змінювати і аналізувати.

Щоб реалізувати патерн Медіатор використовуємо бібліотеку MediatR, яку так само як і попередні бібліотеки завантажуємо з NuGet.

A screenshot of a computer program

Description automatically generated

Рисунок 20 – Встановлення MediatR

Ми можемо сформувати команди, які будуть виконувати функції формування і реалізації бізнес логіки, приймання запитів і формуванню відповідей. Відповіді, в свою чергу, формуємо в своєрідну обгортку, яка буде інформувати чи коректно виконався http-запит, і якщо ні, то яка помилка виникає під час виконання цього запиту.

Сформуємо загальний клас ApiResponseDto:

*public class ApiResponseDto*

*{*

*public string ErrorCode { get; set; }*

*public string Message { get; set; }*

*public bool IsSuccess*

*{*

*get*

*{*

*return string.IsNullOrEmpty(ErrorCode);*

*}*

*}*

*public bool IsCancelled*

*{*

*get*

*{*

*return ErrorCode == "Cancelled";*

*}*

*}*

*}*

Цей клас повертатиме інформацію про сам запит чи коректно він виконався. Для GET запитів нам необхідно також повертати інформацію, для цих випадків використовуємо похідні класи: DataItemApiResponseDto і DataCollectionApiResponseDto

*public class DataItemApiResponseDto<T> : ApiResponseDto*

*where T : BaseItemDto, new()*

*{*

*public T? Item { get; set; }*

*}*

*public class DataCollectionApiResponseDto<T> : ApiResponseDto*

*where T : BaseItemDto, new()*

*{*

*public List<T> Items { get; set; }*

*public int TotalCount { get; set; }*

*}*

Ці «дженерік» класи є загальними для всіх DTO, що наслідковуються від BaseItemDto, відповідно можна використовувати їх, як загальну відповідь.

Розглянемо приклад команди, що повертає список кімнат в розрізі ідентифікатора гуртожитка і номера поверха:

*public class GetRoomItemsCommand : IRequest<DataCollectionApiResponseDto<RoomListItemDto>>*

*{*

*public Guid HostelId { get; set; }*

*public int? Level { get; set; }*

*}*

*public class GetRoomItemsCommandRequestHandler : IRequestHandler<GetRoomItemsCommand, DataCollectionApiResponseDto<RoomListItemDto>>*

*{*

*private readonly RoomService \_roomService;*

*private readonly IMapper \_mapper;*

*private readonly ICacheManager<DataCollectionApiResponseDto<RoomListItemDto>> \_roomCacheManager;*

*public GetRoomItemsCommandRequestHandler(*

*RoomService roomService,*

*IMapper mapper,*

*ICacheManager<DataCollectionApiResponseDto<RoomListItemDto>> roomCacheManager)*

*{*

*\_roomService = roomService;*

*\_mapper = mapper;*

*\_roomCacheManager = roomCacheManager;*

*}*

*public async Task<DataCollectionApiResponseDto<RoomListItemDto>> Handle(GetRoomItemsCommand request, CancellationToken cancellationToken)*

*{*

*try*

*{*

*var entities = await \_roomService.GetByPredicate(c => c.HostelId == request.HostelId && (request.Level == null || c.Level == request.Level.Value));*

*var entitiesDto = entities != null ? entities.Select(c => \_mapper.Map<RoomListItemDto>(c)).ToList() : new();*

*return new DataCollectionApiResponseDto<RoomListItemDto>*

*{*

*Items = entitiesDto*

*};*

*}*

*catch (Exception ex)*

*{*

*return new DataCollectionApiResponseDto<RoomListItemDto>*

*{*

*ErrorCode = "Fail",*

*Message = ex.Message*

*};*

*}*

*}*

*}*

Як тут помітно, обгортаємо всю логіку в try-catch операцію, що дозволить зловити теоретичну помилку і повернути запит на клієнта з текстом цієї помилки і з подальшим опрацюванням цього випадку.

В середині try викликаємо метод GetByPredicate(), що знаходиться в RoomService і передаємо наші умови, що треба отримати список кімнат, які знаходяться в цьому гуртожитку і на цьому поверсі. Всі значення, які підходять під ці умови, будуть записані в змінну entities, що є List<RoomEntity>.

Далі потрібно конвертувати сутності в DTO, які будуть віддані в відповіді на цей http-запит. Для цього потрібно скористатись «маппером», який був описаний вище. За допомогою нього автоматично конвертуємо з RoomEntity в RoomListItemDto, не використовуючи зайвий код.

Після конвертації обгортаємо наш результат в обгортку DataCollectionApiResponseDto<RoomListItemDto> і, якщо опрацювання було успішним, значення IsSuccess буде true.

4.10 Ендпоінти. Бібліотека Ardalis

Таблиці в базах даних створені, сервіси з репозиторієм теж, команди налаштовані, а як зв’язати клієнта з сервером? Для цього нам знадобляться ендпоінти і бібліотека Ardalis.

Ендпоінти - це кінцеві точки доступу (API-кінці), які дозволяють різним додаткам, сайтам, іншим програмним продуктам отримувати доступ до даних, функціональних можливостей або послуг, які пропонує система.

Ендпоінти зазвичай визначаються за допомогою URL-адрес, які включають шлях до ресурсу та формат даних, які буде повернено. Ендпоінти можна використовувати для різних цілей, таких як отримання даних, надсилання даних або виконання дій. Важливо захищати ендпоінти від несанкціонованого доступу, оскільки вони можуть надавати доступ до чутливої інформації, яку не можна поширювати.

Встановимо бібліотеку Ardalis

A screenshot of a computer program

Description automatically generated

Рисунок 21 – Встановлення бібліотеки Ardalis

Після встановлення розглянемо ендпоінт на прикладі отримання даних по кімнатах в розрізі ідентифікатора гуртожитка і номера поверху:

*public class GetRoomItemsEndpoint : BaseAsyncEndpoint*

*{*

*[HttpGet("api/room/items")]*

*[SwaggerOperation(*

*Summary = "Get room items",*

*Description = "Get room items",*

*OperationId = "Room.Items",*

*Tags = new[] { "Room" })*

*]*

*public async Task<ActionResult<DataCollectionApiResponseDto<RoomListItemDto>>> HandleAsync([FromServices] IMediator mediator,*

*[FromQuery] int? level,*

*[FromQuery] Guid hostelId,*

*CancellationToken cancellationToken = new CancellationToken())*

*{*

*var response = await mediator.Send(new GetRoomItemsCommand()*

*{*

*Level = level,*

*HostelId = hostelId,*

*});*

*if (response.IsSuccess)*

*{*

*return Ok(new DataCollectionApiResponseDto<RoomListItemDto>()*

*{*

*Items = response.Items,*

*ErrorCode = response.ErrorCode,*

*Message = response.Message*

*});*

*}*

*else*

*{*

*return BadRequest(new DataCollectionApiResponseDto<RoomListItemDto>()*

*{*

*ErrorCode = response.ErrorCode,*

*Message = response.Message*

*});*

*}*

*}*

*}*

Як можна помітити використовується наслідковування від Ardalis класа BaseAsyncEndpoint, який формує точку входу для GET запиту.

В аннотації передається частина шляху, який є своєрідною адресою входу, так само помічається що це саме GET запит. Параметри level i hostelId очікуємо як частину шляху, які підхоплюються і передаються далі в команду, що будуть своєрідними фільтрами для отримання вірних кімнат.

В середині самого методу викликаємо команду, і передаємо значення ідентифікатору гуртожитка і номера поверху. Після отримання результату віддаємо результат клієнту.

4.11 Middleware

Було розглянуто повний шлях від отримання запиту в ендпоінті і до віддачі сформованої відповіді. Але що стосовно безпеки нашої програми?

Щоб зловмисники не могли користуватись цим софтом було прийнято рішення створити міддлвер, функція якого буде перевірка валідності запиту.

Проміжне́ програ́мне забезпе́чення (скор. ППЗ; англ. middleware; також перекладається як підпрограмне забезпечення, зв'язувальне ПЗ, міжплатфо́рмне ПЗ) — в інформатиці, шар програмного забезпечення, що складається з агентів, які є посередниками між різними компонентами великого застосунка. Найчастіше проміжне ПЗ використовується в розподілених застосунках, причому агентів, що становлять цей шар, може бути кілька.

В проєкті міддлвер виглядає наступним чином:

*public class ApiKeyValidationMiddleware*

*{*

*private readonly RequestDelegate \_next;*

*private readonly IWebHostEnvironment \_env;*

*public ApiKeyValidationMiddleware(RequestDelegate requestDelegate, IWebHostEnvironment env)*

*{*

*\_next = requestDelegate;*

*\_env = env;*

*}*

*public async Task Invoke(HttpContext context)*

*{*

*string token = context.Request.Headers["ApiKey"];*

*var validTokens = new List<string>*

*{*

*"a430a583-dc44-466a-8885-c93fd7822633"*

*};*

*if (token == null || !validTokens.Any(i => i == token))*

*{*

*context.Response.StatusCode = 401;*

*await context.Response.WriteAsync("Access denied.");*

*return;*

*}*

*//pass request further if correct*

*await \_next(context);*

*}*

*}*

Тут перевіряється хедер запиту ApiKey і, якщо він пустий або невірний, повертає 401 Unathorized помилку, в валідному випадку дозволяє пройти до ендпоінта.

Підключення ApiKeyValidationMiddleware в Program.cs:

*app.UseMiddleware<ApiKeyValidationMiddleware>();*

Міддлвер буде працювати перед кожним запитом до сервера, тим самим перевіряючи чи валідний цей запит чи ні.

ВИСНОВКИ

Під час роботи над дипломним проектом було розглянуто низку теоретичних питань:

* поняття поселення в гуртожиток;
* автоматизація процесу поселення в гуртожиток;
* характеристика серверу MSSQL та клієнта Microsoft Visual Studio.

На основі аналізу процесу поселення в гуртожиток було визначено основні функціональні вимоги для автоматизації цього процесу. Для їх реалізації була виконана значна робота з використанням бази даних і бекенд частиною програми. Практична складова дипломної роботи містить:

* розробку бази даних;
* створення сутностей і міграція них;
* розробка репозиторію і сервісів;
* створення команд і бізнес логіки;
* створення ендпоінтів;
* валідація запитів.

Дипломна робота демонструє, як використання знань проєктування серверної частини програми збільшує і автоматизує працездатність роботи, дозволяє більш точно і структуровано вести облік мешканців і кімнат, отримувати дані згідно запитів, додавати і редагувати сутності.

Крім того, дослідження підкреслює важливість впровадження сучасних технологій в управлінні поселення в гуртожиток, що сприяє зменшенню витрат часу та ресурсів на рутинні операції, а також покращує загальну продуктивність та ефективність процесу. Використання автоматизованих систем на основі баз даних допомагає керівництву гуртожитку швидше опрацьовувати облік, забезпечуючи своєчасне поселення і скорочуючи час на цей процес.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Azure https://uk.wikipedia.org/wiki/Microsoft\_Azure
2. Microsoft SQL Server

<https://uk.wikipedia.org/wiki/Microsoft_SQL_Server>

1. Навіщо потрібен automapper? https://forum.itvdn.com/t/zachem-nuzhen-automapper/2846
2. Патерн «Посередник» (Mediator) https://bool.dev/blog/detail/pattern-posrednik-mediator
3. Порівняння СУБД My SQL, PostgreSQL та MS SQL Server

https://data-b-i.com/uk/article/porivnyannya-subd-mysql-postgresql-mssqlserver.htmlAsd

1. Порядок надання жилої площі в гуртожитках. Виселення з гуртожитків

<https://wiki.legalaid.gov.ua/index.php/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BD%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B6%D0%B8%D0%BB%D0%BE%D1%97_%D0%BF%D0%BB%D0%BE%D1%89%D1%96_%D0%B2_%D0%B3%D1%83%D1%80%D1%82%D0%BE%D0%B6%D0%B8%D1%82%D0%BA%D0%B0%D1%85._%D0%92%D0%B8%D1%81%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D0%B7_%D0%B3%D1%83%D1%80%D1%82%D0%BE%D0%B6%D0%B8%D1%82%D0%BA%D1%96%D0%B2>

1. Проміжне програмне забезпечення https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BC%D1%96%D0%B6%D0%BD%D0%B5\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5\_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F
2. Що таке SQL Server Management Studio (SSMS)? https://learn.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16
3. Що таке ЕНДПОІНТИ (endpoints) https://pmtips.com.ua/qna/shcho-take-endpointy