МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Відділ комп'ютерних технологій
Кафедра комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

з обчислювальної практики

з дисципліни «Об'єктно-орієнтоване програмування»

на тему:

«Додаток для готелю»

Студента 2 курсу, групи 244
напряму підготовки «Комп'ютерні науки»
Кульчицький А. С.
(прізвище, ініціали студента)

Керівник доц., к.ф.-м.н. Томка Ю. Я.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала: _____

Кількість балів: _____Оцінка: ECTS ___

Члени комісії:

_____ проф. Ю.О. УШЕНКО
(підпис) (прізвище та ініціали)

_____ доц. Ю.Я. ТОМКА
(підпис) (прізвище та ініціали)

_____ доц. М.Л. КОВАЛЬЧУК
(підпис) (прізвище та ініціали)

Чернівці – 2023

Чернівецький національний університет імені Юрія Федьковича

(назва вузу)

Навчально-науковий інститут фізико-технічних та комп'ютерних наук
Кафедра комп'ютерних наук

ЗАТВЕРДЖУЮ
Зав. кафедрою _____ Ушенко Ю.О.
"19" червня 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на обчислювальну практику

Студента _Кульчицького Андрія Сергійовича _групи_244_другого курсу

(прізвище, ім'я, по батькові)

1. Тема роботи: «Додаток для готелю»

Керівник роботи доц, к.ф.-м.н. Томка Юрій Ярославович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом проекту: **30.06.2023**

3. Вихідні дані до роботи: літературні джерела

об'єктно-орієнтовані підходи до розробки

програмного забезпечення із використанням мови C# у програмному середовищі Microsoft Visual Studio 2022

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити):

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналіз проблематики вивчаємого питання. Опис предметної області.

Аналіз аналогів програмного забезпечення

Функціональні вимоги до розробляємого програмного продукту

Модульно-аналітична схема додатку

База даних

Вибір архітектури розробляємого додатку

РОЗДІЛ 2. РОЗРОБКА ШАРУ РОБОТИ ІЗ ДАНИМИ

Загальний стек використовуваних технологій

Модуль роботи із даними на основі ADO.NET&Dapper

Модуль роботи із даними на основі Entity Framework

Модуль роботи із даними на основі чистої архітектури засобами Entity Framework

РОЗДІЛ 3. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

Концепція MVVM архітектури на рівні вашого клієнтського WPF-додатку

User Flow

VIEWS

Інструкція користувача

VIEWMODELS та COMMANDS

ВИСНОВКИ

5. Перелік наочного матеріалу (з точним зазначенням обов'язкових креслень, плакатів):

- графічна складова користувацького інтерфейсу програмного забезпечення;
- код програми;
- пояснювальна записка;
- презентація.

Дата видачі завдання **19.06.2023**

КАЛЕНДАРНИЙ ПЛАН			
№ з/п	Назва етапів обчислювальної практики	Строк виконання етапів проекту	Примітка
1	Отримання завдання на обчислювальну практику	19.06.2023	
2	Огляд джерел технічної інформації за тематикою	20.06.2023	
3	Виконання аналізу методів реалізації завдання	21.06.2023	
4	Аналіз шляхів рішення поставленої задачі	21.06.2023	
5	Програмна розробка додатку. Модуль роботи із даними на основі ADO.NET&Dapper	21.06.2023	
6	Програмна розробка додатку. Модуль роботи із даними на основі Entity Framework	22.06.2021	
7	Програмна розробка додатку. Модуль роботи із даними на основі чистої архітектури засобами Entity Framework	23.06.2021	
8	Мануальне тестування модулів роботи із даними	24-25.06.2021	
9	Програмна розробка клієнтської частини десктопного додатку засобами WPF	26-28.06.2021	
10	Мануальне тестування розробленого програмного продукту	28.06.2023	
11	Корегування роботи за результатами розгляду керівника. Написання пояснювальної записки.	29.06.2023	
12	Розробка доповіді та презентації	29.06.2023	
13	Захист роботи з обчислювальної практики	30.06.2023	

Студент _____ **Кульчицький А.С.** Керівник _____ **Томка Ю.Я.**

(підпис) (підпис)

Завдання видане «**19**» **червня 2021р.**

АНОТАЦІЯ

Десктоп-додатки забезпечує ефективне та зручне керування готельними операціями та покращення взаємодії з клієнтами. Десктоп-додатки дозволяють готелям автоматизувати багато процесів, таких як управління бронюваннями, розподілом номерів, облік фінансів та інші аспекти їх діяльності.

Задача полягає у тому щоб створити додаток для Windows у якому будуть оголошення про оренду житла та послуги готелю. Потрібно реалізувати можливість створення, видалення та зміни бронювання житла.

Це все буде реалізовано з архітектурним патерном MVVM та за допомогою графічної підсистеми WPF.

Ключові слова: ДЕКСТОП-ДОДАТОК, АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, АРХІТЕКТУРНИЙ ПАТЕРН MVVM, ГРАФІЧНА ПІДСИСТЕМА WPF.

ЗМІСТ

ВСТУП

В сучасному світі існує велика потреба у платформах, які дозволяють різним готелям та власникам житла розміщувати свої об'яви і надавати можливість орендувати їх користувачам. Це дозволяє забезпечити зручну інтерактивну платформу для бронювання житла на певну дату, на заданий період часу та вибір категорії.

За допомогою цього готелю легче збільшувати кількість клієнтів та це полегшує процес оренди кімнати. Користувачі можуть шукати житло за певною категорією, вибирати дату, яка їм підходить, і орендувати його на зазначений період часу.

Метою роботи є проектування і розробка платформи для оренди житла.

Для досягнення мети поставлено такі **завдання:**

– проаналізувати предметну область та аналоги ПЗ;
– сформулювати функціональні вимоги до системи;
– продумати структуру проекту відповідно до вимог;
– реалізувати back-end додатку на платформі .NET;
– реалізувати front-end з використання технології WPF.

Структура та обсяг роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (1 найменування), Загальний обсяг роботи становить 28 сторінок, серед них 4 рисунків.

РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ

1.1 Формування проблематики. Опис предметної області

Інтернет відіграє важливу роль у готельному бізнесі, забезпечуючи широкі можливості для реклами, взаємодії з клієнтами та надання послуг. Платформа для оренди житла створює зручні умови для клієнтів, які шукають житло на певну дату та тривалість. Це дає їм можливість ознайомитись з типами кімнат, їхньою ціною та послугами який надає готель.

Платформа дозволяє клієнтам швидко та зручно знайти і орендувати житло, а також обирати за ціновими категоріями, Це забезпечує комфортний процес вибору для клієнтів, оскільки вся інформація легко доступна та може бути легко сортована та фільтрована.

Крім того, додаток для оренди житла може забезпечувати додаткові функції, такі як онлайн-оплата, відгуки та рейтинги, список з послуг, місце знаходження готелю та контактна інформація. Це полегшує процес оренди та покращує взаємодію між клієнтами та готелем.

Загалом, додаток готелю сприяє ефективному використанню ресурсів, забезпечує зручність та широкі можливості для користувачів. Вона створює вигоду як для клієнтів, які шукають житло, так і для орендодавців, які пропонують свої об'єкти оренди.

Для цієї предметної області потрібно реалізувати:

1.Бронювання, їх створення та редагування клієнтами.

2. Реалізація відгуків та системи рейтингу.

3. Пошук з фільтрами оголошень за критеріями.

4. Система реєстрації, автентифікації та авторизації користувачів.

5. Профіль користувача.

6. Зворотній зв'язок та контактна інформація.

1.2 Аналіз аналогів програмного забезпечення

Аналіз аналогів ПЗ дасть змогу виділити сильні сторони, які можна перейняти, та з’ясувати слабкі сторони які не потрібно реалізовувати у своєму додатку. Як аналоги є такі платформи як Booking та Airbnb.

1.2.1 Rivoli boutique hotel

Rivoli boutique hotel – це чотирьох зірковий готель у Італії, розташований в комфортному туристичному маленькому містечку

Сильною стороною цього сайту є простий та інтуїтивно зрозумілий дизайн для користувача , Він мінімалістичний і в ньому легко орієнтуватись , всі важливі елементи мають яскравий колір,

Також розробники подбали про різноманітність мов, це дозволило збільшити кількість клієнтів серед туристів з інших країн.

Рисунок 1.1 – Користувацький інтерфейс готелю Rivoli boutique

1.2.2 Підсумки аналізу

Під час аналізу ми виділили сильні сторони сайту, це допоможе зрозуміти як створити сайт з комфортним дизайном та для людини з будь якої країни. Застосувавши їх на практиці можливо збільшти кількість клієнтів та прибуток готелю.

1.3 Функціональні вимоги до проекту. User Stories

User Stories (історії користувача) – це спосіб представлення функціональних вимог до програмного забезпечення на етапі проектування; набір коротких речень простою мовою предметної області, які чітко описують потреби користувача у функціоналі. Вони поширені у розробці ПЗ. Окрема

історія користувача – з’ясовує, хто має функціональну вимогу, що вона собою являє та чому її необхідно реалізувати

Можна виділити такі типи: Анонім, Читач, Автор, Адміністратор. Кожен наступний тип наділений всім функціоналом попереднього та новим.

Далі наведено перелік всіх User Stories, сформованих для розробки проекту. Деякі з них було відібрано для подальшого розвитку; такі історії позначено «НР» (не реалізовано).

1.3.1 User Stories Аноніма

1. Як анонім, я хочу переглядати всі варіанти оренди від найновішої до найстарішої, щоб отримати необхідну мені інформацію.

2. Як анонім, я хочу мати посторінковий перегляд пропозицій, щоб заощаджувати ресурси мого комп’ютера при завантаженні вебсайту.

3. Як анонім, я хочу бачити перелік найбільш популярних відгуків, щоб знати, на що варто звернути увагу. (НР)

4. Як анонім, я хочу бачити перелік категорій, щоб мати уявлення про наявність необхідного мені контенту. (НР)

5. Як анонім, я хочу фільтрувати кімнати по категоріях, щоб пришвидшити пошук необхідної публікації. (НР)

6. Як анонім, я хочу мати довідкову інформацію про вебсайт, щоб швидко та доступно ознайомитися з ним. (НР)

7. Як анонім, я хочу отримати контакти власника вебсайту, щоб могти зв’язатися з ним зі скаргами та пропозиціями. Бажано мати вбудовану форму зворотного зв’язку. (НР)

8. Як анонім, я хочу обирати мою мову зі списку, щоб вільно розуміти вміст вебресурсу. При цьому повинно бути перекладено все, окрім вмісту користувачів (публікацій, коментарів, даних профілю). (НР)

9. Як анонім, я хочу реєструватися в системі, щоб створити власний обліковий запис із додатковими привілеями. (HP)

10. Як анонім, я хочу входити в систему зі своїми даними, щоб отримувати доступ до власного облікового запису. (HP)

11. Як анонім, я бажаю доступних та чітких повідомлень про неправильно введені дані, щоб швидко розуміти, в чому я допустив помилку. (HP)

12. Як анонім, я хочу бачити в публікації не тільки суцільний текст, а й форматування, зображення, медіа. Це допоможе більш наочно донести мені інформацію. (HP)

13. Як анонім, я хочу бачити середню користувацьку оцінку публікації, щоб орієнтуватися в її релевантності та змістовності. (HP)

1.3.2 User Stories Читача

1. Як читач, я хочу мати доступ до налаштувань власного профілю, де я міг би редагувати особисті дані та вподобання, щоб мати контроль над моєю взаємодією з додатком. (HP)

2. Як читач, я хочу залишати коментарі до публікацій, щоб висловлювати свою думку щодо них. (HP)

3. Як читач, я хочу відповідати на коментарі, щоб вести спілкування із конкретним користувачем. При цьому він повинен бути повідомлений про мою відповідь. (HP)

4. Як читач, я хочу бачити власну історію переглядів номерів, щоб повертатися до переглянутої інформації в разі потреби. (HP)

5. Як читач, я хочу отримувати необхідні оновлення у вигляді електронних листів, щоб слідкувати за виходом цікавих мені публікацій. (HP)

1.3.3 User Stories Автора

1. Як автор, я хочу мати перелік власних публікацій з інструментами для роботи з ними, щоб зручно проводити свою діяльність на ресурсі. (HP)

2. Як автор, я хочу створювати публікацію на єдиній формі, включно з додаванням категорій та зображення, щоб зробити швидшим процес публікування. (HP)

3. Як автор, я хочу редагувати публікацію, зокрема замінювати зображення заставки, щоб могти виправляти свої помилки. (HP)

4. Як автор, я хочу мати попередній перегляд створюваної або редагованої публікації, щоб мати уявлення про зовнішній вигляд результату роботи. (HP)

5. Як автор, я повинен мати посібник по розмітці в публікації, щоб навчитися грамотно організовувати її вміст. (HP)

6. Як автор, я хочу відображати власні публікації в моєму профілі, щоб інші користувачі могли мати легкий доступ до них. (HP)

7. Як автор, я хочу мати загальний рейтинг на вебсайті, базований на оцінках моїх публікацій, щоб давати уявлення іншим про мою діяльність. (HP)

1.3.4 User Stories Адміністратора

1. Як адміністратор, я хочу оперативно видаляти будь-який коментар, щоб уникнути неприємних ситуацій. (HP)

2. Як адміністратор, я хочу керувати користувачами, зокрема призупиняти дію облікових записів, щоб обмежити їх у разі необхідності. (HP)

3. Як адміністратор, я хочу робити публікації прихованими, щоб їх автори могли виправити критичні проблеми. (HP)

4. Як адміністратор, я хочу мати перелік вхідних заявок на роль автора, щоб зручно і оперативно на них відповідати. (HP)

5. Як адміністратор, я хочу мати перелік потенційно небажаних коментарів, щоб оперативно реагувати на порушення. (HP)

6. Як адміністратор, я хочу отримувати статистику відвідуваності сайту, щоб робити судження про аудиторію та бізнес-рішення. (HP)

1.4 Проектування бази даних

Для реалізації сутностей предметної області обрано реляційну Microsoft SQL Server. На зображені створення бази данних за допомогою CodeFirst підходу:

```
24 references
public class HotelSystemContext : DbContext
{
    0 references
    public HotelSystemContext(DbContextOptions<HotelSystemContext> options)
        : base(options)
    {
    }

    0 references
    public DbSet<Amenity> Amenities { get; set; }
    0 references
    public DbSet<Booking> Bookings { get; set; }
    0 references
    public DbSet<BookingService> BookingServices { get; set; }
    0 references
    public DbSet<Payment> Payments { get; set; }
    0 references
    public DbSet<Review> Reviews { get; set; }
    0 references
    public DbSet<Room> Rooms { get; set; }
    0 references
    public DbSet<RoomAmenity> RoomAmenities { get; set; }
    0 references
    public DbSet<Service> Services { get; set; }
    0 references
    public DbSet<User> Users { get; set; }
```

Рисунок 1.2 – Контекст данних

Важливим є візуалізація таблиць та їх відношень один з одним, ось ERD бази данних нашого додатку для Windows

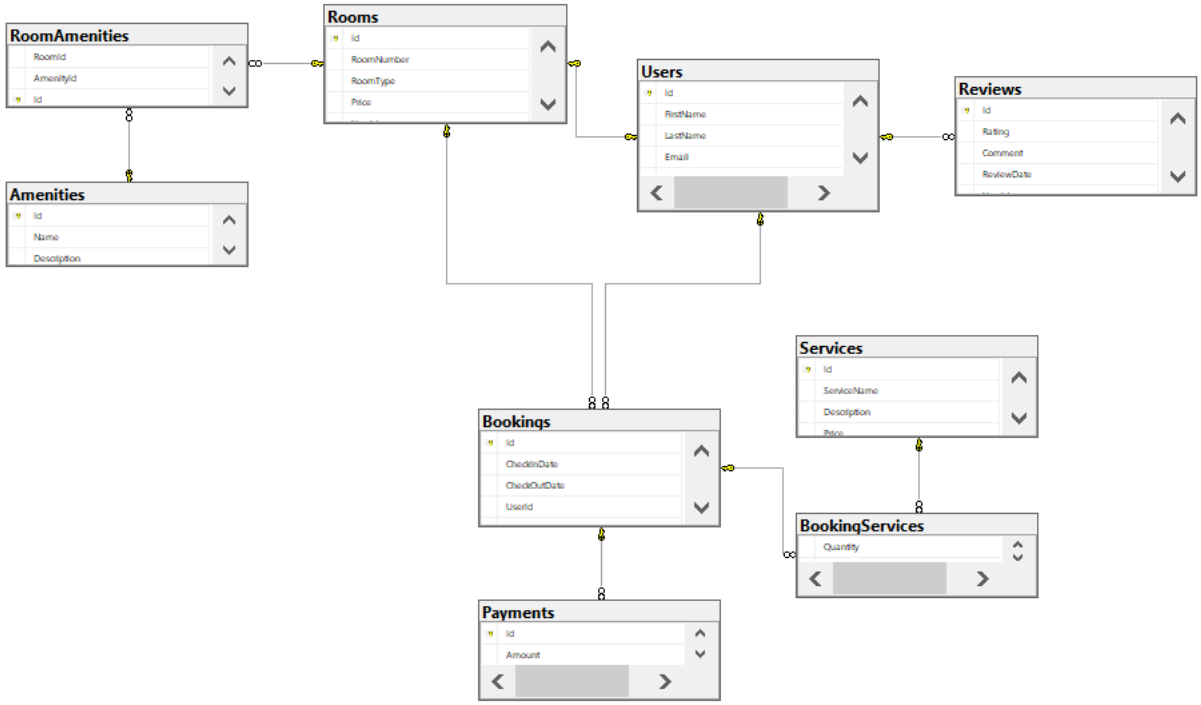

Рисунок 1.5 – Діаграма створеної бази даних

1. Bookings (Записник) – це основна сутність, у якісь ведуться записи про оренду житла, замовлення послуг та їх оплати, пов’язана з іншими важливими таблицями такі як Users та Rooms

2. Users (Користувачі) – одна з основних сутностей, яка відповідає за користувачів з різним рівнем можливостей . Містить його повне ім’я пароль та псевдонім для індитифікації у додатку, пошту для відновлення данних чи контакту з власником

3. Rooms (Кімнати) ця сутність відображає кімнати, їхню ціну та предмети побуту, повязується з користувачем який її орендує та записником, є дата початку та кінця оренди

1.4 Вибір архітектури додатку

Вимога проекту це використання архітектурного патерну MVVM.

Model-View-ViewModel [1]— шаблон проєктування, що застосовується під час проєктування архітектури застосунків (додатків). Публічно вперше був

представлений Джоном Госсманом (John Gossman) у 2005 році як модифікація шаблону Presentation Model. MVVM орієнтований на такі сучасні платформи розробки, як Windows Presentation Foundation та Silverlight від компанії Microsoft.

MVVM полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі). Модель представлення є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. З цієї точки зору, модель представлення більше схожа на модель, ніж на представлення і оброблює більшість, якщо не всю, логіку відображення даних. Модель представлення може також реалізовувати патерн медіатор, організовуючи доступ до бек-енд логіки навколо множини правил використання, які підтримуються представленням. Шаблон MVVM ділиться на три частини:

Модель (Model), як і в класичному шаблоні MVC, Модель являє собою фундаментальні дані, що необхідні для роботи застосунку.

Вигляд (View) як і в класичному шаблоні MVC, Вигляд — це графічний інтерфейс, тобто вікно, кнопки тощо.

Модель вигляду (ViewModel, що означає «Model of View») з одного боку є абстракцією Вигляду, а з іншого надає обгортку даних з Моделі, які мають зв'язуватись. Тобто вона містить Модель, яка перетворена до Вигляду, а також містить у собі команди, якими може скористатися Вигляд для впливу на Модель. Фактично ViewModel призначена для того, щоб

- Здійснювати зв'язок між моделлю та вікном
- Відслідковувати зміни в даних, що зроблені користувачем
- Відпрацьовувати логіку роботи View (механізм команд)

ВИСНОВКИ ДО РОЗДІЛУ

У цьому розділі було обрано предментну область а саме додаток для готелю, описано її проблематику, проведено аналіз аналогу Rivoli boutique та розглянуто плюси які поліпшать створюваний додаток, та попередять помилки яких припустиились їх розробники.

Сформовано перелік User Stories, що визначають повний список функціональних вимог, у тому числі тих, що будуть реалізовані в майбутньому. Визначено вимоги для таких типів користувачів системи: Анонім, Читач, Автор, Адміністратор.

Спроектовано первинну базу даних засобами MS SQL Server за допомогою CodeFirst підходу. Наведено структуру та тлумачення сутностей і зв’язків у базі даних.

Розглянуто архітектуру MVVM, її особливості, плюси та мінуси

РОЗДІЛ 2. BACK-END РОЗРОБКА

1.1 Вибір стеку технологій

Стек технологій, що був використаний для створення десктоп додатку для готелю, включає такі компоненти: .NET, WPF, EF Core Fluent API та Generic Host. Ось опис кожної з цих технологій та причини їх вибору:

.NET (або .NET Framework/.NET Core): .NET - це розвиток мови програмування C#, який надає широкий набір бібліотек, середовища виконання та інструментів для розробки програмного забезпечення. Використання .NET дозволяє розробникам створювати потужні, масштабовані та надійні додатки для різних платформ, включаючи десктоп. .NET також має багатий екосистему, включаючи широкий вибір сторонніх бібліотек і фреймворків.

WPF (Windows Presentation Foundation): WPF є фреймворком для створення графічних інтерфейсів користувача в десктоп додатках для Windows. Використання WPF дозволяє розробникам створювати багатофункціональні та привабливі за дизайном додатки з допомогою декларативного підходу до UI розмітки та використання стилів, шаблонів та анімацій. WPF також надає можливості для прив'язки даних та розміщення елементів у вікні з допомогою гнучких контейнерів.

EF Core Fluent API (Entity Framework Core): EF Core - це сучасна технологія ORM (Object-Relational Mapping), яка надає простий спосіб взаємодії з базами даних у десктоп додатках. Fluent API в EF Core дозволяє визначати модель даних та налаштовувати відображення між моделлю та базою даних з допомогою покрокових методів. Використання EF Core спрощує роботу з базою даних, забезпечує безпеку, швидкість та можливість масштабування.

Generic Host: Generic Host - це механізм, який надає загальний шаблон для створення хосту додатка, який може включати різні компоненти, такі як сервіси, конфігурацію та журналування. Використання Generic Host спрощує розгортання та управління додатком, забезпечує більшу гнучкість та розширюваність. Вибір Generic Host для десктоп додатка може покращити його архітектуру та роботу з різними модулями.

Вибір цього стеку технологій для розробки десктоп додатку для готелю має наступні переваги:

.NET надає розробникам потужні інструменти та різноманітні бібліотеки для швидкої та ефективної розробки програмного забезпечення.

WPF дозволяє створювати привабливий та функціональний графічний інтерфейс користувача.

EF Core Fluent API спрощує роботу з базою даних та забезпечує швидкий доступ до даних.

Використання Generic Host дозволяє покращити архітектуру додатка та забезпечити його гнучкість та розширюваність.

Цей стек технологій є популярним в галузі розробки десктоп додатків та дозволяє розробникам ефективно створювати потужні та привабливі додатки для готелів.

1.2 Основна реалізація з використанням Entity Framework Core

1.3.1 Code-First підхід до створення бази даних

Code-First підхід – це комфортна методика для створювання бази данних, коли використовується EF core

Кожна сутність у додатку має набір спільних полів – ідентифікатор, дат створення да останньої зміни. Нижче наведено код абстрактної сутності ModelBase, яка є базовим класом для інших.

```
public abstract class ModelBase
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
    }
```

Ось приклади конкретних сутностей : Post, Comment. Решта сутностей реалізовані схожим чином і повторюють структуру бази даних (див. підрозділ 1.5). Окрім властивостей, що повторюють поля таблиці, є також навігаційні властивості, які використано для конфігурації зв'язків між сутностями.

```
public class Review : ModelBase
    {
        [Required]
        public double Rating { get; set; }

        public string? Comment { get; set; }

        [Required]
        [DataType(DataType.Date)]
        public DateTime? ReviewDate { get; set; }


        //connections
        public int UserId { get; set; }
        public User User { get; set; } = null!;
    }


public class Room : ModelBase
    {
        [Required]
        public int RoomNumber { get; set; }

        [Required]
        public string RoomType { get; set; } = null!;

        [Required]
        public double Price { get; set; }


        //connections
        public int UserId { get; set; }
        public User User { get; set; } = null!;

        public ICollection<Booking> Bookings { get; set; } = new List<Booking>();
       public ICollection<RoomAmenity> RoomAmenities { get;set; } = new
List<RoomAmenity>()
       }
```

Для моделювання таблиць в Entity Framework Core –присутній контекстний клас DbContext. Він містить набір властивостей типу DbSet, що моделюють таблиці в базі даних. У додатку повинен бути власний клас контексту, що наслідується від DbContext. Ось оголошення моделей які будуть реалізовані у бд

```
        public HotelSystemContext(DbContextOptions<HotelSystemContext> options)
            : base(options)
        {
        }

        public DbSet<Amenity> Amenities { get; set; }
        public DbSet<Booking> Bookings { get; set; }
        public DbSet<BookingService> BookingServices { get; set; }
        public DbSet<Payment> Payments { get; set; }
        public DbSet<Review> Reviews { get; set; }
        public DbSet<Room> Rooms { get; set; }
        public DbSet<RoomAmenity> RoomAmenities { get; set; }
```

```
public DbSet<Service> Services { get; set; }
public DbSet<User> Users { get; set; }
}
```

Наведеного вище коду достатньо, щоб EF Core міг створити таблиці без звязків та з набором полів. Однак для накладання обмежень та ключів також застосовується конфігурування контексту.

1.3.2 Конфігурація сутностей з використанням Fluent API

Сутності можна конфігурувати або безпосередньо в методі OnModelCreating, або створити класи конфігурації для кожної сутності та винести логіку в них, що дозволяє розділити обов'язки та зменшити об'єм контексту. Так реєструються в OnModelCreating всі класи конфігурацій зі збірки:

```
public void Configure(EntityTypeBuilder<User> builder)
        {
            builder.HasKey(u => u.Id);

            builder.Property(u => u.FirstName)
                .HasMaxLength(25);

            builder.Property(u => u.LastName)
                .HasMaxLength(25);

            builder.Property(u => u.Email)
                .HasMaxLength(50);

            builder.Property(u => u.Username)
                .HasMaxLength(50);

            builder.Property(u =>u.Password)
                .HasMaxLength(32);


            //connections
            builder.HasMany(u => u.Reviews)
                .WithOne(r => r.User)
                .HasForeignKey(r => r.UserId);

            builder.HasMany(u => u.Bookings)
                .WithOne(b => b.User)
                .HasForeignKey(b => b.UserId)
                .OnDelete(DeleteBehavior.NoAction);

            builder.HasOne(u => u.Room)
                .WithOne(r => r.User)
                .HasForeignKey<Room>(r => r.UserId);


            builder.HasData(
            new User
            {
```

```
            Id = 1,
            FirstName = "John",
            LastName = "Doe",
            Email = "john.doe@example.com",
            Username = "johndoe",
            Password = "password1",
            RoomId = 1
        },
        new User
        {
            Id = 2,
            FirstName = "Jane",
            LastName = "Smith",
            Email = "jane.smith@example.com",
            Username = "janesmith",
            Password = "password2",
            RoomId = 2
        },
        new User
        {
            Id = 3,
            FirstName = "Mike",
            LastName = "Johnson",
            Email = "mike.johnson@example.com",
            Username = "mikejohnson",
            Password = "password3",
            RoomId = 3
        },
        new User
        {
            Id = 4,
            FirstName = "Sarah",
            LastName = "Williams",
            Email = "sarah.williams@example.com",
            Username = "sarahwilliams",
            Password = "password4",
            RoomId = 4
                    });
```

За допомогою Fluent API можна створювати звязки між таблицями, створювати назви таблиць, та заповнювати їх. В трьох останніх ланцюгах методів можна побачити заповнення.

Ось приклад підключення конфігураційних файлів до кнтексту

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.ApplyConfiguration(new AmenityConfiguration());
            modelBuilder.ApplyConfiguration(new BookingConfiguration());
            modelBuilder.ApplyConfiguration(new BookingServiceConfiguration());
            modelBuilder.ApplyConfiguration(new PaymentConfiguration());
            modelBuilder.ApplyConfiguration(new ReviewConfiguration());
            modelBuilder.ApplyConfiguration(new RoomAmenityConfiguration());
            modelBuilder.ApplyConfiguration(new RoomConfiguration());
            modelBuilder.ApplyConfiguration(new ServiceConfiguration());
            modelBuilder.ApplyConfiguration(new UserConfiguration());
        }
```

1.3.3 Generic Repository. Unit of Work

У поєднанні з Entity Framework Core, Generic Repository по суті виступає додатковою абстракцією над DbSet, щоб послабити залежність. Разом із ним також використовується підхід Unit of Work («одиниця роботи») – передача об’єкту, що містить посилання на всі конкретні репозиторії, і є абстракцією над DbContext. Таким чином, через Unit of Work можна отримати доступ до будь-яких потрібних даних. Крім того, він відкидає потребу в оновленні окремих репозиторіїв, оскільки сам повинен містити метод збереження всіх змін. У всіх цих патернах велику роль відіграють інтерфейси.

Конкретні репозиторії наслідують клас Generic Repository, тобто мають стандартний набір методів, а також надають власні, що стосуються певної більш точкової роботи з даною сутністю. Наприклад, ось коли всі операції CRUD були імплементовані у Generic Repository:

```
public class UserRepository : GenericRepository<User>, IUserRepository
{
    public UserRepository(HotelSystemContext context) : base(context)
    { }
      }
```

Тут за допомогою технології LINQ to Entities відбувається робота з сутностями через DbSet, а також застосовано метод Include для eager loading («жадібного завантаження») зв’язаних сутностей через навігаційні властивості.

Для використання отриманих класів необхідно зареєструвати їх для dependency injection у конфігурації проекту ASP.NET Core:

```
builder.Services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));
        builder.Services.AddScoped(typeof(IGenericTransitiveReposetory<>),
typeof(GenericTransitiveRepository<>));
        builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();
        builder.Services.AddScoped<IAmenityService, AmenityService>();
        builder.Services.AddScoped<IBookingServices, BookingServices>();
        builder.Services.AddScoped<IBookingServiceService, BookingServiceService>();
        builder.Services.AddScoped<IPaymentService, PaymentService>();
        builder.Services.AddScoped<IReviewService, ReviewService>();
        builder.Services.AddScoped<IRoomService, RoomService>();
        builder.Services.AddScoped<IRoomAmenityService, RoomAmenityService>();
        builder.Services.AddScoped<IServiceService, ServiceService>();
                    builder.Services.AddScoped<IUserService, UserService>();
```

1.3.4 Міграції. Зміни структури бази даних у процесі розробки

Міграція один із важливих моментів у CodeFirst, адже За допомогою механізму міграцій EF Core дозволяє ітеративно оновлювати структуру базу даних, залежно від змін у сутностях та їх конфігурації. У консолі менеджера пакетів, встановивши пакет EntityFrameworkCore.Tools, можна ввести команди:

1. *add-migration MigrationName* – створює нову міграцію з усіма незбереженими змінами структури бази даних.

2. *update-database* – оновлює структуру базу даних відповідно до останньої міграції.

1.3.5 Бізнес-логіка. DTO. AutoMapper

У шарі бізнес-логіки ми отримуємо доступ до даних через ін’єкцію Unit of Work до сервісів, та з його використанням формуємо потрібні методи. Приклад бізнес-логіки (створення публікації у UserService):

```
public class UserService : IUserService
    {
        private readonly IUnitOfWork _uow;
        private readonly IMapper _mapper;

        public UserService(IUnitOfWork unitOfWork, IMapper mapper)
        {
            _uow = unitOfWork;
            _mapper = mapper;
        }


        public async Task<IEnumerable<UserDTO>> GetAllAsync()
        {
            var result = await _uow.User.GetAllAsync();
            return _mapper.Map<IEnumerable<User>, IEnumerable<UserDTO>>(result);
        }


        public async Task<UserDTO> GetByIdAsync(int Id)
        {
            var result = await _uow.User.GetByIdAsync(Id);

            return _mapper.Map<UserDTO>(result);
        }

        public async Task<UserDTO> CreateAsync(PostUserDTO userDTO)
        {
            var result = await _uow.User.CreateAsync(_mapper.Map<User>(userDTO));

            return _mapper.Map<UserDTO>(result);
        }
```

```
        public async Task<UserDTO> UpdateAsync(UserDTO userDTO)
        {
            var result = await _uow.User.UpdateAsync(_mapper.Map<User>(userDTO));

            return _mapper.Map<UserDTO>(result);
        }

        public async Task<UserDTO> DeleteAsync(int Id)
        {
            var result = await _uow.User.DeleteAsync(Id);

            return _mapper.Map<UserDTO>(result);
                    }
```

Метод приймає дані користувача, створена публікація повертається з методу.

Тут, очевидно, стикаємося з потребою перетворення форми даних у процесі роботи додатку. Так, дані, що надходять до сервісу ззовні, відрізняються від структури сутностей, аналогічно й дані, що сервіс видає назовні. Для таких цілей використовуються класи DTO (Data Transfer Object – об'єкт транспортування даних). У даному проекті вони поділені на дві групи: вхідні дані (Requests, або запити) та вихідні (Responses, або відповіді).

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace HotelSystem_EF.Bll.DTO.User
{
    public class UserDTO : INotifyPropertyChanged
    {
        public int Id { get; set; }
        private string _firstName;
        private string _lastName;
        private string _email;
        private string _username;
        private string _password;


        [Required]
        public string FirstName
        {
            get { return _firstName; }
            set
            {
                _firstName = value;
                OnPropertyChanged(nameof(FirstName));
            }
        }

        [Required]
        public string LastName
        {
            get { return _lastName; }
            set
            {
```

```
                _lastName = value;
                OnPropertyChanged(nameof(LastName));
            }
        }

        [Required]
        public string Email
        {
            get { return _email; }
            set
            {
                _email = value;
                OnPropertyChanged(nameof(Email));
            }
        }
        [Required]
        public string Username
        {
            get { return _username; }
            set
            {
                _username = value;
                OnPropertyChanged(nameof(Username));
            }
        }

        [Required]
        public string Password
        {
            get { return _password; }
            set
            {
                _password = value;
                OnPropertyChanged(nameof(Password));
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        public virtual void OnPropertyChanged([CallerMemberName] string prop = "")
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(prop));
        }
    }
}
```

Задачу перетворення даних із сутності в DTO та навпаки виконує бібліотека AutoMapper (потребує реєстрації сервісів для ін’єкції). Вона надає набір методів та способи конфігурації «маппінгів». Приклад нижче:

```
private void UserMapperConfigurations()
        {
            CreateMap<User, UserDTO>();

            CreateMap<UserDTO, User>();
            CreateMap<PostUserDTO, User>();
              }
```

Дана конфігурація вказує, за яким правилом перетворювати ті чи інші властивості. За замовчуванням значення копіюються в однойменні властивості, проте інші випадки необхідно налаштовувати. Так як це маппінг для створення, тут немає індетифікатора, адже він створюється автоматично при кожному ствоерннi сутності

ВИСНОВКИ ДО РОЗДІЛУ
РОЗДІЛ 3. FRONT-END РОЗРОБКА

3.1 Концепція MVVM архітектури на рівні вашого клієнтського WPF-додатку

MVVM (Model-View-ViewModel) - це архітектурний патерн, який використовується для розробки додатків з графічним інтерфейсом користувача. Він дозволяє розділити логіку бізнес-логіки додатку від його представлення та взаємодії з користувачем. Основні компоненти MVVM включають модель (Model), представлення (View) та модель представлення (ViewModel). Крім цього, для ефективного взаємодії зі в'юшками, можуть використовуватись команди (Commands), структури даних (Data Bindings) та інші елементи.

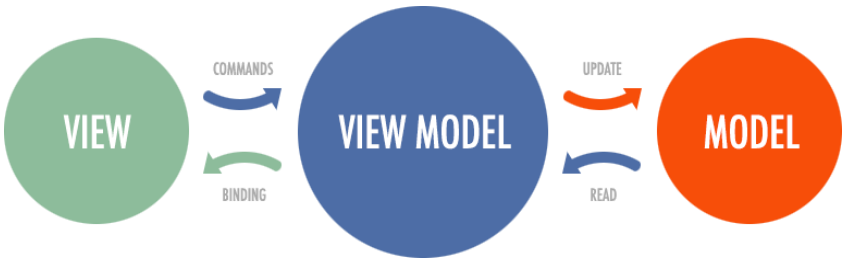

Рисунок 3.6 – Схема MVVM патерну

Ось розширений опис компонентів MVVM:

Модель (Model):
Модель представляє бізнес-логіку та дані додатку.
Вона може містити класи, які відповідають за отримання/збереження даних, валідацію, обробку логіки тощо.
Модель не повинна містити жодної залежності від представлення або моделі представлення.

Представлення (View):
Представлення відповідає за відображення графічного інтерфейсу користувача.
Це може бути XAML-файл або кодовий шматок, який визначає структуру та вигляд елементів інтерфейсу.
Представлення може використовувати прив'язки даних (Data Bindings) для отримання даних з моделі представлення.

Модель представлення (ViewModel):
Модель представлення є проміжним шаром між моделлю та представленням.
Вона містить логіку, яка підготовлює дані з моделі для відображення в представленні.
Модель представлення використовує прив'язки даних для зв'язку з представленням та оновлення його відповідно до змін у моделі.

Команди (Commands):
Команди дозволяють виконувати певну дію на представленні під час взаємодії користувача.
Команди реалізуються в моделі представлення і можуть бути пов'язані з елементами керування, такими як кнопки або контекстні меню.
Вони надають можливість відділити логіку взаємодії від відображення.

Прив'язки даних (Data Bindings):
Прив'язки даних дозволяють зв'язувати властивості елементів представлення з властивостями моделі представлення.
Це дозволяє автоматично оновлювати відображення відповідно до змін в моделі представлення та передавати введені користувачем дані назад в модель.

Ця структура MVVM дозволяє забезпечити відокремлення логіки, простоту тестування та повторне використання коду. Кожен компонент має свою відповідальність і може бути легко модифікований чи замінений без впливу на інші частини додатку. Крім того, використання команд та прив'язок даних спрощує взаємодію між користувачем і додатком, забезпечуючи більш динамічні та реагуючі інтерфейси.

3.2 View

View є компонентом, який відповідає за візуальне представлення даних та інтеракцію з користувачем. View відображає дані з ViewModel і реагує на події, що виникають в користувацькому інтерфейсі.

Основні характеристики та функції View в MVVM патерні:

- Представлення даних: View відображає дані, які надаються ViewModel. Це може бути текст, зображення, списки або будь-які інші візуальні елементи, які відображають стан додатку або деякі модельні дані.

- Прив'язка даних: View використовує механізми прив'язки даних для отримання даних з ViewModel та автоматичного оновлення візуальних елементів при зміні даних у ViewModel. Це дозволяє забезпечити актуальність відображених даних і уникнути прямої залежності між View та ViewModel.

Рисунок 3.7 – Представлення данних, які надійшли з ViewModel

- Обробка подій: View реагує на події, що виникають в інтерфейсі користувача, такі як натискання кнопок, введення тексту або переміщення миші. Ці події передаються до ViewModel для подальшої обробки логіки та оновлення модельних даних.

3.3 ViewModel

ViewModel[2] реалізує властивості та команди, до яких подання може прив’язувати дані, і сповіщає подання про будь-які зміни стану через події сповіщень про зміни. Властивості та команди, які надає модель перегляду, визначають функціональні можливості, які пропонуватиме користувальницький інтерфейс, але представлення визначає, як ці функції мають відображатися.

ViewModel є важливою складовою патерну MVVM (Model-View-ViewModel) і використовується для відокремлення логіки бізнес-логіки від користувацького інтерфейсу.

Основна мета ViewModel полягає в тому, щоб забезпечити зручну та ефективну взаємодію між моделлю даних (Model) і представленням (View).

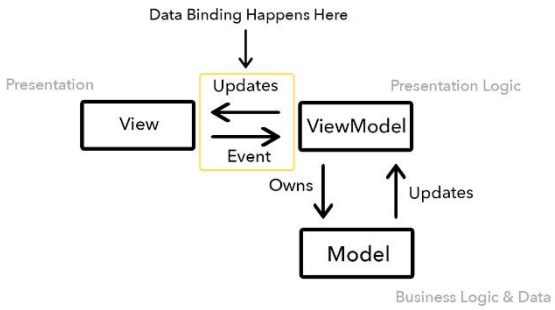

Рисунок 3.8 – Ілюстрація взаємодії за допомогою MVVM

Для того, щоб модель перегляду брала участь у двосторонньому зв’язуванні даних із представленням, його властивості повинні викликати подію PropertyChanged. Моделі перегляду задовольняють цю вимогу, реалізуючи інтерфейс INotifyPropertyChanged і викликаючи подію PropertyChanged, коли властивість змінено.

```
11 references
public class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    22 references
    protected virtual void OnPropertyChanged(string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    6 references
    protected virtual void Dispose() { }
}
```

Рисунок 3.9 – Реалізація базового класу для ViewModel з інтерфейсом INotifyPropertyChanged

ВИСНОВОК ДО РОЗДІЛУ

MVVM (Model-View-ViewModel) є архітектурним патерном, який дозволяє розділити бізнес-логіку додатку від його представлення та взаємодії з користувачем. Основні компоненти MVVM включають модель (Model), представлення (View) та модель представлення (ViewModel). Модель відповідає за бізнес-логіку та дані, представлення відображає графічний інтерфейс, а модель представлення забезпечує взаємодію між моделлю та представленням. Для ефективної взаємодії можуть використовуватись команди, прив'язки даних та інші елементи. MVVM дозволяє забезпечити відокремлення логіки, простоту тестування та повторне використання коду. View відповідає за візуальне представлення даних та інтеракцію з користувачем, використовуючи прив'язку даних та обробку подій. ViewModel відокремлює бізнес-логіку від інтерфейсу та забезпечує зручну взаємодію з моделлю даних та представленням.

ВИСНОВОК

В ході розробки декстоп додатку для готелю було проведено аналіз проблематики та предметної області, досліджено наявні рішення на ринку та виділено їх сильні та слабкі сторони. Були визначені функціональні вимоги до програмного продукту і розлоблені user stories які планується реалізувати.

В рамках модульно-аналітичної схеми була вказана модульна структура додатку, а також була наведена схема бази даних з описом таблиць та їх полів. Для підключення до бази даних була використана концепція EF (EntityFramework) з генерічним репозиторієм та Unit Of Work Також було розроблено шар бізнес-логіки, включаючи сервіси, валідування та маппінг даних.

У клієнтській частині була використана архітектура MVVM (Model-View-ViewModel) з окремими класами для VIEW та VIEWMODEL. Були розроблені команди для взаємодії між VIEW та VIEWMODEL. Завдяки Dependency Injection та Generic Host було забезпечено підняття додатку.

ДЖЕРЕЛА

1. Model-View-ViewModel – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Model-View-ViewModel
2. ViewModel in MVVM – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm