

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПІДТРИМКИ РОБОТИ
ГОТЕЛЬНОГО СЕРВІСУ
DEVELOPMENT OF THE APPLICATION FOR
THE HOTEL SERVICE SUPPORT

Виконав(ла): студент(ка) 4 курсу, групи КНТ-139

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КУШНІРЕНКО Д.А.

(ПРІЗВИЩЕ та ініціали)

Керівник ШИТІКОВА О.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КУШНІРЕНКА Данила Андрійовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка застосунку для підтримки роботи готельного сервісу . Development of the Application for the Hotel Service Support

керівник проєкту (роботи) к.т.н., ШИТІКОВА Олена Вікторівна,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 04 ” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз ринку застосунків для готельного бізнесу. 2. Вибір мови програмування та технологій розробки. 3. Проєктування застосунку для підтримки готельного сервісу. 4. Розробка та реалізація хмарного застосунку для підтримки готельного сервісу.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ШИТІКОВА О.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Данило КУШНІРЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Олена ШИТІКОВА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 113 с., 3 табл., 27 рис., 2 дод., 56 джерел.

ГОТЕЛЬ, ГОТЕЛЬНИЙ СЕРВІС, ГОТЕЛЬНИЙ МЕНЕДЖЕР, VISUALSTUDIO, C++, MICROSOFTAZURE, MICROSOFTDEVOPS.

Об'єкт дослідження – процес керування готелями.

Предмет дослідження – розробка застосунку для підтримки готельного сервісу.

Мета роботи – розробка застосунку, що може спростити та автоматизувати різні готельні операції, такі як бронювання кімнат, реєстрація гостей, облік оплати, керування запасами тощо.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мови програмування C++, MicrosoftAzure, MicrosoftDevops, VisualStudio, персональний комп'ютер з процесором з частотою 2.5 ГГц під управлінням операційної системи Microsoft Windows 10.

Результати. створено хмарний застосунок, що дозволяє здійснювати перегляд, додавання, редагування та видалення інформації про бронювання, рахунки гостей готелю, співробітників готелю, товари та послуги, що використовуються в готелі.

Висновки. Розроблено програмний застосунок який значно облегшує роботу готельного сервісу.

Галузь використання – готельний сервіс.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
113 pages, 3 tables, 27 figures, 2 appendix, 56 sources.

HOTEL, HOTEL SERVICE, HOTEL MANAGER, VISUAL STUDIO,
C++, MICROSOFT AZURE, MICROSOFT DEVOPS.

The object of research is the hotel management process.

The subject of the research is the development of an application for hotel service support.

The purpose of the work is to develop an application that can simplify and automate various hotel operations such as room reservations, guest registration, payment accounting, inventory management, etc.

Materials, methods and technical means: structural and object-oriented programming, C++ programming languages, Microsoft Azure, Microsoft Devops, Visual Studio, a personal computer with a 2.5 GHz processor running the Microsoft Windows 10 operating system.

The results. created a cloud-based application that allows you to view, add, edit and delete information about reservations, hotel guest accounts, hotel employees, goods and services used in the hotel.

Conclusions. A software application has been developed that greatly facilitates the work of the hotel service.

The field of use is hotel service.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз ринку застосунків для готельного бізнесу	11
1.1 Типи програмних забезпечень для керування готелем	11
1.1.1 Хмарне програмне рішення	11
1.1.2 Внутрішнє готельне рішення	12
1.2 Аналіз програмних аналогів	12
1.2.1 АСУHotelogix	13
1.2.2 АСУ OracleHospitality	15
1.2.3 АСУ FrontdeskAnywhere.....	17
1.2.4 ПорівнянняАСУ	19
1.3 Постановка завдання.....	20
2 Вибір мови програмування та технологій розробки	21
2.1 Обґрунтування вибору хмарного програмного рішення	21
2.1.1 Хмарна платформа Azure	22
2.1.2 Технології хмарного обчислення	22
2.2 Обґрунтування вибору мови програмування.....	26
2.3 Обґрунтування інструменту розробки.....	27
3 Проєктування застосунку для підтримки готельного сервісу	29
3.1 Функціональні вимоги до проєкту	29
3.2 Архітектура системи	30
3.3 Структура бази даних	32
4 Розробка та реалізація хмарного застосунку для підтримки готельного сервісу	38
4.1 Реалізаціязастосунку мовою С++	38
4.1.1 Управління бронюваннями	40
4.1.2 Управління номерами	41
4.1.3 Управління рахунками	42

4.1.4 Управління персоналом.....	43
4.1.5 Управління запасами	44
4.2 Реалізація хмарного кросплатформеного застосунку	45
4.3 Особливості використання застосунку	46
4.4 Особливості взаємодії з користувачем	47
4.5 Перевірки реалізовані під час виконання коду	49
4.6 Відлагодження та тестування розробленого сервісу, порівняння результатів роботи з існуючими аналогами.....	50
Висновки	51
Перелік джерел посилань	52
Додаток А Текст програми.....	58
Додаток Б Слайди презентації	107

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

АСУ – автоматизована система управління;

ПЗ – програмне забезпечення;

СУБД – системи управління базами даних.

ВСТУП

В сучасному світі оптимізація робочих процесів є надзвичайно актуальною для компаній. Кожна організація прагне підвищити продуктивність своєї роботи, впроваджуючи передові технології. З розквітом туризму в нашій країні, готельний сервіс став однією з ключових галузей обслуговування. На сьогоднішній день в готельній сфері існує велика конкуренція, що змушує впроваджувати найсучасніші та найкомфортніші технології в системи управління готелями. Самі системи управління готелями досить складні, оскільки вони складаються з багатьох взаємопов'язаних служб та підсистем.

У сучасному готельному бізнесі важливо бути в курсі новітніх технологій і використовувати їх для розвитку підприємства, інакше можна поступитися місцем конкурентам. Одним з основних інструментів для досягнення успіху в цій галузі є використання спеціалізованого програмного забезпечення, яке допомагає автоматизувати робочі процеси, встановлювати ефективні взаємини з клієнтами, керувати персоналом, створювати фінансові документи, проводити аналітику і маркетинг.

З огляду на постійні вимоги клієнтів до якості обслуговування, готелі все частіше впроваджують сучасні технології в свої системи підтримки клієнтів. Це можуть бути рішення для гостьових кімнат, системи онлайн-бронювання, системи аудіо та відео зв'язку та інші. Такі системи спрямовані на полегшення для організацій виявлення, приваблення, розвитку та підтримки вигідних взаємин з клієнтами, забезпечуючи високу якість обслуговування та створюючи лояльних клієнтів.

Проте деякі готелі досі не мають достатнього потенціалу для впровадження таких технологій та поліпшення взаємодії з клієнтами. Вони можуть бути позбавлені необхідного апаратного та програмного забезпечення, яке дозволяє здійснювати онлайн-бронювання, керувати бронюванням номерів, відстежувати клієнтів та підвищувати ефективність

готельних процесів. В результаті, такі готелі не можуть підтримувати актуальну базу даних своїх клієнтів.

Отже, актуальною задачею є проведення аналізу та розробка оптимальної системи управління готелем.

Об'єкт дослідження – процес керування готелями.

Предмет дослідження – розробка застосунку для підтримки готельного сервісу.

Мета роботи – розробка застосунку, що може спростити та автоматизувати різні готельні операції, такі як бронювання кімнат, реєстрація гостей, облік оплати, керування запасами тощо.

Для досягнення цієї мети необхідно виконати наступні завдання:

- провести аналіз предметної області та розглянути аналогічні застосунки підтримки готельного сервісу;
- зробити вибір мови програмування та інших технологій розробки;
- розробити архітектуру застосунку;
- реалізувати застосунок підтримки готельного сервісу, який буде містити функції управління бронюваннями, номерами, рахунками, персоналом та запасами.

1 АНАЛІЗ РИНКУ ЗАСТОСУНКІВ ДЛЯ ГОТЕЛЬНОГО БІЗНЕСУ

1.1 Типи програмних забезпечень для керування готелем

У минулому, програмне забезпечення для управління готелями мало обмежений доступ і встановлювалося локально на комп'ютерах у фізичному приміщенні готелю. Це означало, що доступ до програми був можливий лише з цих комп'ютерів і не був доступний з будь-якого іншого місця. Однак, з появою хмарних рішень, стало можливим отримати доступ до програмного забезпечення та керувати ним з будь-якого місця, де є Інтернет-підключення.

На сьогоднішній день існують як локальні, так і хмарні рішення для управління готелями, які мають відмінності в функціональних можливостях та способах доступу.

Локальні рішення передбачають встановлення програмного забезпечення на комп'ютерах у фізичному приміщенні готелю. Такі системи надають можливість керувати бронюваннями, обліком клієнтів, фінансами та іншими аспектами управління готелем. Прикладами локальних програмних рішень є "Hotelogix" [1] та "Opera PMS" [2].

Хмарні рішення забезпечують доступ до програмного забезпечення через Інтернет. Вони розміщені на віддалених серверах та доступні для використання з будь-якого пристрою з Інтернет-підключенням. Хмарні системи дозволяють здійснювати керування готелем, включаючи бронювання, управління номерами, фінансами та звітністю. Прикладами хмарних рішень є "Cloudbeds" [3] та "Hotello" [4].

1.1.1 Хмарне програмне рішення

Хмарні готельні рішення пропонують інтуїтивно зрозумілі та легкі в освоєнні інтерфейси, що дозволяють вашому персоналу зосередитися на задоволенні потреб гостей, а не на боротьбі з надмірно складним або непотрібним програмним забезпеченням [5]. З розвитком галузі все більше

готелів переходять на хмарні програмні рішення для готелів, і розробники постійно додають нові функціональні можливості.

Хмарні рішення систем управління готелями мають низку переваг, які притаманні хмарному програмному забезпеченню взагалі. Ви зможете отримати повний доступ до своєї системи з будь-якого місця, де є доступ до веббраузера. Оскільки програмне забезпечення розташоване в хмарі, ви завжди матимете останню версію програми, і ваша ІТ-команда не потребує встановлення оновлень. Наявність останньої версії гарантує вам відповідність місцевому законодавству та вимогам забезпечення відповідності [6].

1.1.2 Внутрішнє готельне рішення

Протягом тривалого часу внутрішнє програмне забезпечення для готелів існує і постійно вдосконалюється, щоб задовольнити зростаючі потреби готельної галузі. Цей процес постійного вдосконалення може призводити до розширення функціоналу, що іноді ускладнює його використання, особливо для співробітників, які не мають досвіду роботи з цим конкретним програмним забезпеченням [7].

На протязі багатьох років внутрішнє програмне забезпечення для готелів постійно вдосконалюється, щоб відповідати змінним потребам галузі. Цей процес еволюції часто супроводжується додаванням нових функцій, які можуть бути складними у використанні, особливо для співробітників, які не мають попереднього досвіду роботи з цим програмним забезпеченням [8].

1.2 Аналіз програмних аналогів

Перед розробкою застосунку для підтримки роботи готельного сервісу важливо проаналізувати існуючі програмні аналоги, які виконують схожі

функції. Ось кілька прикладів програмних аналогів, які можуть бути використані для порівняння [9]-[11]:

- **Hotelogix**. Це хмарна система управління готельним бізнесом, яка надає функціональність для управління бронюваннями, номерами, рахунками, персоналом та запасами. Вона пропонує широкий спектр можливостей, таких як онлайн-бронювання, звітність, фактурування та інтеграцію з іншими системами. Однак, вона може бути обмежена в налаштуванні та розширенні функціональності;

- **Oracle Hospitality**. Ця система управління готелем включає модулі для бронювань, фінансів, ресторанного бізнесу, персоналу та інших аспектів готельної діяльності. Вона надає розширені можливості для інтеграції з іншими системами та забезпечує високий рівень безпеки даних. Однак, вона може бути складною у використанні та вимагати значних витрат на ліцензії та підтримку;

- **FrontdeskAnywhere**. Це хмарна система управління готелем, яка пропонує функціональність для бронювань, номерів, рахунків та персоналу. Вона має зручний інтерфейс користувача, дозволяє керувати бронюваннями з будь-якого пристрою та надає звітність для аналізу даних. Проте, вона може бути менш гнучкою в порівнянні з іншими аналогами.

Аналізуючи ці програмні аналоги, можна визначити їх переваги та недоліки, які можуть бути використані для покращення розроблюваного застосунку. При розробці застосунку є можливість створити власну унікальну систему, яка відповідає конкретним потребам готельного сервісу, забезпечуючи гнучкість, швидкодію та безпеку даних.

1.2.1 АСУ Hotelogix

Hotelogix [10] є хмарною системою управління готельним бізнесом, яка надає широкий спектр функцій для підтримки роботи готельних закладів.

Основними функціями Hotelogix є управління бронюваннями, номерами, рахунками, персоналом та запасами.

Деякі з ключових особливостей цієї системи включають [10]:

- управління бронюваннями. Hotelogix дозволяє менеджерам готелю легко переглядати, додавати, редагувати та видаляти бронювання. Вона також забезпечує реальний час перегляду наявності вільних номерів і надсилає сповіщення про нові бронювання та нагадування про наближення дати заїзду;

- управління номерами. Hotelogix дозволяє ефективно керувати інформацією про кімнати, включаючи тип кімнати, ціни, доступність та інші деталі. Вона також надсилає сповіщення про нові бронювання, які вимагають нової кімнати, та нагадування про потребу в ремонті та обслуговуванні кімнат;

- управління рахунками. За допомогою Hotelogix менеджери готелю можуть легко переглядати, додавати, редагувати та видаляти рахунки гостей. Система дозволяє створювати рахунки за розміщення, харчування, додаткові послуги та інші витрати. Вона також надсилає сповіщення про нові рахунки та нагадування про заборгованість з боку гостей;

- управління персоналом. Hotelogix надає можливість менеджерам готелю переглядати, додавати, редагувати та видаляти інформацію про співробітників готелю, включаючи дані про персонал, обов'язки та графіки роботи. Це сприяє ефективному плануванню графіків роботи, розподілу обов'язків та забезпеченню потрібної кількості персоналу для задоволення потреб гостей;

- управління запасами. Hotelogix дозволяє менеджерам готелю переглядати, додавати, редагувати та видаляти інформацію про товари та послуги, що використовуються в готелі. Ця функція полегшує контроль за запасами, замовленнями та управлінням постачальниками.

Hotelogix надає готелям потужний інструментарій для ефективного управління різними аспектами готельного бізнесу, полегшуючи роботу менеджерів та покращуючи задоволення гостей.

Інтерфейс застосунку HOTELOGIX зображено на рисунку 1.1.



Рисунок 1.1 – Інтерфейс HOTELOGIX [10]

Хоча Hotelogix є потужною системою для управління готельним бізнесом, варто враховувати, що кожен готель має свої унікальні потреби та вимоги. При розробці власного застосунку є можливість налаштувати його точно під свої потреби та внести необхідні зміни, щоб максимально задовольнити потреби вашого готельного сервісу.

1.2.2 ACU OracleHospitality

OracleHospitality [11] є комплексною платформою управління готельним бізнесом, розробленою компанією Oracle. Вона надає широкий спектр функціональних можливостей для підтримки роботи готельних закладів будь-якого розміру і типу.

Основні характеристики та функціонал OracleHospitality включають [11]:

- управління бронюваннями. OracleHospitality дозволяє готелям ефективно керувати процесом бронювання. Вона надає можливість

переглядати, додавати, редагувати та видаляти бронювання, контролювати наявність вільних номерів та встановлювати ціни. Крім того, система може автоматично надсилати повідомлення про нові бронювання та нагадування про наближення дати заїзду;

- управління номерами. OracleHospitality дозволяє готелям керувати всією інформацією про номери. Менеджери можуть легко додавати, редагувати та видаляти дані про номери, включаючи типи номерів, розташування, розміри та зручності. Система також може надсилати повідомлення про нові бронювання, які вимагають певного типу номера, а також нагадування про необхідність ремонту та обслуговування номерів;

- управління рахунками. OracleHospitality дозволяє готелям легко створювати, переглядати, редагувати та видаляти рахунки гостей. Система автоматично розраховує вартість проживання, харчування, додаткових послуг та інших витрат. Вона також може надсилати повідомлення про нові рахунки та нагадування про заборгованість з боку гостей;

- управління персоналом. OracleHospitality допомагає готелям управляти персоналом. Система дозволяє менеджерам переглядати, додавати, редагувати та видаляти інформацію про співробітників готелю, включаючи дані про особові дані, обов'язки та графіки роботи. Вона також може допомагати в плануванні графіків роботи, розподілі обов'язків та управлінні доступом персоналу;

- управління запасами. OracleHospitality надає інструменти для ефективного управління запасами в готелі. Менеджери можуть переглядати, додавати, редагувати та видаляти інформацію про товари та послуги, що використовуються в готелі. Система допомагає відстежувати запаси, робити замовлення та контролювати постачання.

OracleHospitality є відомим постачальником рішень для готельного бізнесу та має великий досвід у галузі готельного сервісу. Ця платформа надає широкий функціонал і може бути налаштована під вимоги конкретного готеля. Важливо враховувати, що OracleHospitality є комерційним продуктом,

і його використання може вимагати відповідних ліцензій і витрат на розгортання та підтримку.

Інтерфейс застосунку ORACLEHOSPITALITY зображено на рисунку 1.2.

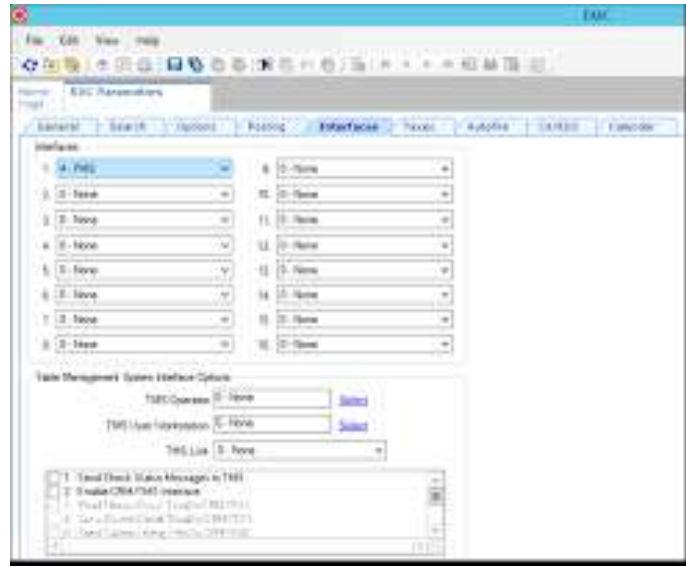


Рисунок 1.2 – Інтерфейс Oracle Hospitality [11]

1.2.3 ACY Frontdesk Anywhere

Frontdesk Anywhere [9] - це хмарне програмне забезпечення, яке допомагає готелям управляти своїм бізнесом. Воно надає широкий спектр функціональних можливостей для ефективного керування бронюваннями, кімнатами, клієнтами та фінансами.

Основні функції FrontdeskAnywhere включають [9]:

- управління бронюваннями. Готелі можуть легко додавати, редагувати та скасовувати бронювання через систему FrontdeskAnywhere. Вона також надає інформацію про гостей та їх контакти, спрощуючи комунікацію;

- управління кімнатами. Готелі можуть переглядати графіки кімнат, додавати та редагувати інформацію про кожну кімнату, щоб ефективно керувати їх зайнятістю;

- управління клієнтами. Система дозволяє готелям створювати профілі клієнтів з інформацією про контакти, попередні перебування та інші дані, що сприяють персоналізації обслуговування;
- фінансове управління. FrontdeskAnywhere допомагає готелям відстежувати доходи, витрати, керувати рахунками та платежами, а також створювати звіти та аналізувати фінансові дані;
- звіти та аналітика. Система надає інструменти для створення звітів та аналізу різних даних, включаючи інформацію про кімнати, бронювання, фінанси та інше;
- мобільні застосунки. FrontdeskAnywhere забезпечує можливість керувати бізнес-операціями готелю з мобільного пристрою, що забезпечує зручність та доступність незалежно від місця знаходження.

FrontdeskAnywhere пропонує зручний та інтуїтивно зрозумілий інтерфейс, який дозволяє готелям легко впроваджувати та використовувати систему без особливих зусиль. Вона також гарантує безпеку даних та регулярні оновлення для покращення функціональності та продуктивності [9].

Інтерфейс застосунку FRONTDESKANYWHERE зображено на рисунку 1.3.

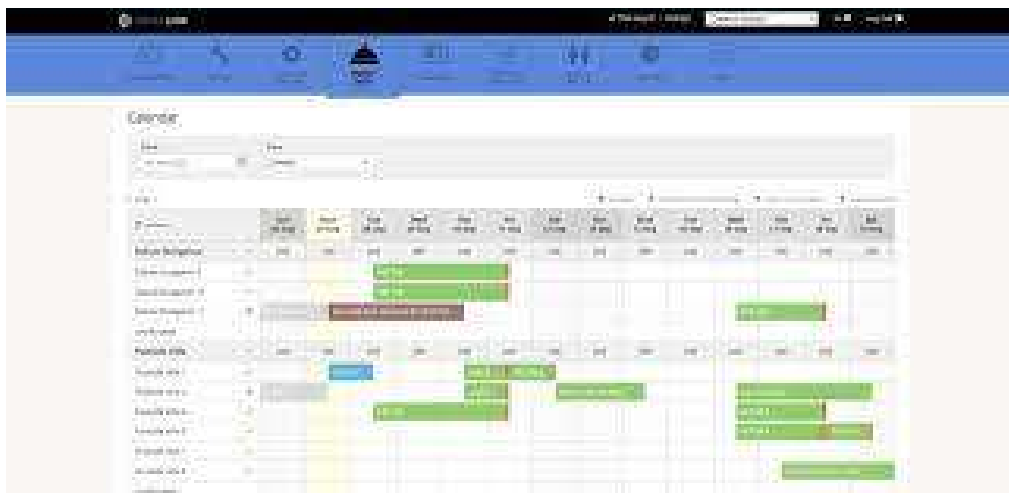


Рисунок 1.3 – Інтерфейс Frontdesk Anywhere [9]

1.2.4 Порівняння АСУ

В таблиці 1.1 представлено порівняння трьох вищенаведених АСУ.

Таблиця 1.1 - Порівняння платформ

Функції/Платформи	Frontdesk Anywhere	Hotelogix	Oracle Hospitality
Управління бронюваннями	+	+	+
Управління номерами	+	+	+
Управління рахунками	+	+	+
Управління персоналом	+	-	+
Управління запасами	-	+	+
Хмарна платформа	+	+	+
Інтуїтивний інтерфейс	+	+	+
Мобільні застосунки	+	+	-
Інтеграція з іншими системами	-	+	+
Вартість	-	+	-
Функціональність	-	-	+

Для розробки програмного застосунку, що підтримує готельний бізнес з функціями управління бронюваннями, управління номерами, управління рахунками, управління персоналом та управління запасами, існує можливість використовувати різні платформи та розробницькі інструменти.

Порівнюючи такі системи автоматизованого управління готелем, як Frontdesk Anywhere [9], Hotelogix [10] і Oracle Hospitality [12], можна знайти багато корисної інформації щодо їх функціональності, надійності та придатності для готельного бізнесу. Кожна з цих платформ дозволяє зручно переглядати, додавати, редагувати та видаляти інформацію про бронювання, номери, рахунки, персонал та запаси. Крім того, системи автоматично надсилають сповіщення про нові бронювання, нагадування про наближення дати заїзду, а також про необхідність ремонту та технічного обслуговування кімнат. Таким чином, розробка програмного застосунку з подібними функціями є цілком доцільною.

1.3 Постановка завдання

Метою дипломної кваліфікаційної роботи є розробка застосунку, який надасть підтримку готельному сервісу, забезпечуючи виконання наступних функцій [13]-[15]:

- управління бронюваннями. Система бронювань для гостей готелю, де менеджери зможуть переглядати, додавати, редагувати та видаляти бронювання. Крім того, система автоматично надсилатиме сповіщення про нові бронювання та нагадування про наближення дати заїзду;

- управління номерами. Система управління кімнатами готелю, де менеджери зможуть переглядати, додавати, редагувати та видаляти інформацію про кімнати. Крім того, система автоматично надсилатиме сповіщення про нові бронювання, які потребують нової кімнати, а також нагадування про необхідність ремонту та технічного обслуговування кімнат;

- управління рахунками. Система управління рахунками гостей, де менеджери зможуть переглядати, додавати, редагувати та видаляти рахунки. Крім того, система автоматично надсилатиме сповіщення про нові рахунки та нагадування про заборгованість з боку гостей;

- управління персоналом. Система управління персоналом готелю, де менеджери зможуть переглядати, додавати, редагувати та видаляти інформацію про співробітників готелю, їхні обов'язки та графіки роботи;

- управління запасами. Система управління запасами готелю, де менеджери зможуть переглядати, додавати, редагувати та видаляти інформацію про товари та послуги, які використовуються в готелі.

Ці функції допоможуть забезпечити ефективну організацію роботи готелю та покращити задоволення гостей під час їх перебування. Для розробки такого застосунку можна використати різні платформи та розробницькі інструменти.

2 ВИБІР МОВИ ПРОГРАМУВАННЯ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Обґрунтування вибору хмарного програмного рішення

Вибір хмарного рішення для розробки готельного застосунку має численні переваги, які можна обґрунтувати наступними факторами:

- гнучкість та масштабованість. Хмарні рішення, наприклад Amazon Web Services (AWS) [16] та Microsoft Azure [17], забезпечують гнучкість та можливість легкого масштабування застосунку, що дає змогу розширювати його функціональні можливості та обсяги при збільшенні обсягу даних та кількості користувачів;

- доступність та зручність. Хмарні рішення дозволяють працювати з застосунком з будь-якого місця, де є доступ до Інтернету. Наприклад, Google Cloud Platform (GCP) [18] надає зручний доступ до застосунку через веббраузер, що сприяє спільній роботі та ефективному управлінню готельними процесами;

- оновлення та безпека. Використання хмарного рішення дозволяє мати постійний доступ до оновлень та покращень програмного забезпечення. Провідні хмарні платформи, такі як AWS, Azure та GCP, забезпечують регулярні оновлення та забезпечують високий рівень безпеки системи;

- інтеграція та співпраця. Хмарні рішення дозволяють легко інтегрувати готельний застосунок з іншими системами та сервісами. Наприклад, застосунок може бути інтегрований з системами бронювання, фінансовими платформами та іншими сервісами для поліпшення обміну даними та ефективного управління готельним бізнесом.

Таким чином, використання хмарного рішення для розробки готельного застосунку дозволяє забезпечити гнучкість, доступність, безпеку та можливості інтеграції, що сприяють ефективному управлінню готельним бізнесом і задоволенню потреб клієнтів.

2.1.1 Хмарна платформа Azure

Azure [17] - це хмарна платформа, призначена для спрощення процесу створення сучасних додатків. Вона надає можливість розміщення програм повністю в Azure або розширення локальних програм за допомогою служб Azure. Azure допомагає розробникам створювати масштабовані, надійні та підтримувані програми. Завдяки широкій підтримці інструментів, таких як VisualStudio та VisualStudioCode, а також повної бібліотеки SDK, Azure сприяє продуктивній розробці.

Сучасні додатки та платформи хмарних обчислень швидко розвиваються в усіх галузях, змінюючи способи функціонування бізнесу та спрощуючи процеси. За даними, сьогодні близько 77% компаній мають свою обчислювальну інфраструктуру в хмарі [19].

На ринку існує багато платформ хмарних обчислень, проте дві з них виявляються особливо домінуючими. AmazonWebServices (AWS) і MicrosoftAzure - це два головні гравці у світі хмарних обчислень. Хоча AWS є найбільшою платформою хмарних обчислень, MicrosoftAzure є найшвидше зростаючою і є другою за величиною [20].

2.1.2 Технології хмарного обчислення

Хмарні обчислення [21] - це технологія, що надає доступ до різних обчислювальних ресурсів через Інтернет. Для використання цієї технології вам потрібно лише підключитися до свого хмарного провайдера за допомогою комп'ютера або мобільного пристрою через Інтернет. Після підключення ви отримуєте доступ до обчислювальних ресурсів, таких як безсерверні обчислення, віртуальні машини, сховище та інші.

Зараз постачальники хмарних послуг, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), мають великі центри обробки даних [22]. Ці центри обробки даних складаються з сотень серверів,

систем зберігання та компонентів, що забезпечують надійну та ефективну роботу. Вони розташовані в безпечних місцях і зберігають великі обсяги даних.

Користувачі підключаються до цих центрів обробки даних, щоб отримувати дані або використовувати їх за потреби. Хмарні послуги надають різноманітні можливості, включаючи отримання сповіщень про нові текстові повідомлення або електронну пошту [23]. Однією з головних переваг хмарних платформ є те, що ви платите лише за користування послугами, без передоплати.

Хмарні обчислення знайшли широке застосування в різних галузях. Вони використовуються для машинного навчання, аналізу даних, зберігання та резервного копіювання, потокового медіа та багатьох інших цілей [21]. Наприклад, платформа Netflix використовує хмарні обчислення для зберігання і стримінгу своїх шоу та фільмів [24]. Крім того, хмарні обчислення дозволяють створювати та тестувати програми, автоматизувати поставку програмного забезпечення та розміщувати блоги [25].

Microsoft Azure є одним з провідних постачальників хмарних послуг і надає більше 200 послуг, які можна розділити на 18 категорій [26]. Ці категорії включають обчислення, мережі, сховище, Інтернет речей, міграцію, мобільні пристрої, аналітику, контейнери, штучний інтелект та машинне навчання, інтеграцію, інструменти керування, інструменти розробника, безпеку, бази даних, DevOps, медіа-ідентифікацію та вебсервіси [26].

Обчислювальні послуги:

- віртуальні машини (Virtual Machines) [27]. Сервіс Azure Virtual Machines дозволяє створювати віртуальні машини з різними операційними системами, такими як Windows або Linux, з миттєвим розгортанням. Ви можете налаштувати віртуальну машину згідно з ваших потреб та працювати з нею, як звичайним комп'ютером;

- хмарні програми (Cloud Services) [28]. Сервіс Azure Cloud Services дозволяє створювати та розгортати масштабовані програми в хмарному

середовищі. Azure бере на себе надання, балансування навантаження та моніторинг працездатності програм, що дозволяє вам зосередитись на розробці програмного забезпечення;

- сервісна структура (Service Fabric) [29]. Azure Service Fabric спрощує розробку та керування мікросервісами. Ви можете створювати компоненти програми, які підтримують відокремлене функціонування та можуть містити інші пакети програм;

- функції (Functions) [30]. Сервіс Azure Functions дозволяє вам створювати програми безпосередньо за допомогою коду. Вам не потрібно виконувати налаштування обладнання, оскільки Azure надає необхідне середовище для виконання ваших функцій.

Мережа:

- мережа доставки вмісту (Azure CDN) [31]. Azure CDN (Content Delivery Network) дозволяє швидко доставляти вміст користувачам за допомогою розподіленої мережі серверів, розташованих по всьому світу. Це допомагає забезпечити швидкий доступ до вмісту для користувачів з різних локацій;

- експресний маршрут (ExpressRoute) [32]. Сервіс Azure ExpressRoute дозволяє підключити вашу локальну мережу до хмари Microsoft або інших хмарних платформ через приватне з'єднання. Це забезпечує приватний та безпечний обмін даними між вашою корпоративною мережею та хмарними сервісами;

- віртуальна мережа (Virtual Network) [33]. Azure Virtual Network дозволяє різним службам Azure комунікувати приватно та безпечно одна з одною. Ви можете налаштовувати віртуальні мережі для ізоляції та керування мережевим трафіком в хмарному середовищі;

- Azure DNS [34]. Сервіс Azure DNS дозволяє розміщувати ваші домени DNS або системні домени в Azure. Ви можете керувати DNS-записами та налаштуваннями для ефективного управління вашими доменами.

Зберігання:

- зберігання диску (Disk Storage) [35]. Azure Disk Storage дозволяє вам обирати між жорстким диском (HDD) та твердотілим накопичувачем (SSD) для зберігання даних, що використовуються разом з вашою віртуальною машиною;
- зберігання блобів (Blob Storage) [36]. Azure Blob Storage є оптимізованим сервісом для зберігання великої кількості неструктурованих даних, таких як текстові або бінарні файли;
- зберігання файлів (File Storage) [37]. Azure File Storage - це керована служба зберігання файлів, до якої можна отримати доступ за допомогою протоколу Server Message Block (SMB). Вона надає спільний доступ до файлів для різних комп'ютерів та серверів;
- зберігання черги (Queue Storage) [38]. Azure Queue Storage дозволяє створювати стійку чергу повідомлень для обробки великого робочого навантаження. Ця служба дає змогу забезпечити послідовну та надійну доставку повідомлень і доступ до неї можна отримати з будь-якої точки світу.

Використання Azure:

- розробка додатків. Ви можете створити будь-яку вебпрограму в Azure;
- тестування. Після успішної розробки програми на платформі ви можете протестувати її;
- розміщення програм. Після завершення тестування Azure може допомогти вам розмістити програму;
- створення віртуальних машин. Ви можете створювати віртуальні машини будь-якої конфігурації за допомогою Azure;
- функції інтеграції та синхронізації. Azure дозволяє інтегрувати та синхронізувати віртуальні пристрої та каталоги;
- збирайте та зберігайте показники. Azure дає змогу збирати та зберігати показники, які допоможуть знайти те, що працює;

– віртуальні жорсткі диски. Це розширення віртуальних машин; вони забезпечують величезний обсяг зберігання даних.

2.2 Обґрунтування вибору мови програмування

Вибір мови програмування для розробки застосунку для підтримки роботи готельного сервісу залежить від різних факторів, таких як функціональні вимоги, вимоги до продуктивності, наявність ресурсів та вміння розробників.

Порівняння мов програмування представлені в таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Мова програмування	Переваги	Недоліки
1	2	3
C++	- висока продуктивність і швидкодія.	- складніша розробка та підтримка.
	- низькорівневий доступ до ресурсів.	- обмежена екосистема готових рішень.
	- можливість оптимізації коду.	
	- підтримка багато потокового програмування.	
Python	- простий та зрозумілий синтаксис.	- помірна продуктивність.
	- багата екосистема готових рішень та бібліотек.	
	- велика кількість доступної документації.	
	- підтримка скриптового програмування.	
Java	- переносимість між платформами.	- вищі вимоги до ресурсів.
	- велика екосистема готових рішень та бібліотек.	
	- підтримка багатопотокового програмування.	
	- сильна типізація та об'єктно-орієнтована парадигма.	

Продовження таблиці 2.1

1	2	3
Ruby	- елегантний та зручний синтаксис.	- помірна продуктивність.
	- багата екосистема готових рішень та фреймворків.	
	- підтримка функціонального та об'єктно-орієнтованого програмування.	
	- зручність розробки та підтримки коду.	

Незважаючи на те, що всі мови, що представлені в таблиці 2.1, мають свої переваги та можуть бути використані для розробки застосунків для підтримки готельного сервісу, C++ виділяється своєю високою продуктивністю та швидкодією [39]. Як низькорівнева мова програмування, C++ дозволяє безпосередньо керувати ресурсами та оптимізувати код для досягнення кращої продуктивності. Оскільки програмні застосунки для готельного сервісу часто потребують обробки великих обсягів даних та виконання складних операцій, використання C++ може бути особливо вигідним з точки зору продуктивності та швидкодії.

2.3 Обґрунтування інструменту розробки

Вибір Microsoft DevOps для проєкту має наступні обґрунтування [40]:

- інтегрована платформа. Microsoft DevOps є повноцінною інтегрованою платформою, що об'єднує всі необхідні інструменти для розробки, тестування, доставки та управління життєвим циклом програмного забезпечення. Це дозволяє зменшити необхідність використовувати різні інструменти та спрощує управління процесом розробки [41];

- спільна робота команди. Microsoft DevOps надає зручні інструменти для спільної роботи команди розробників, що полегшує обмін ідеями, швидке виявлення проблем та спільне розроблення рішень [42];

- автоматизація процесів. Microsoft DevOps надає можливості для автоматизації повторюваних процесів розробки та доставки програмного забезпечення, що дозволяє зосередитися на розробці якісного коду та вирішенні складних завдань [43];

- інтеграція з хмарними сервісами. Microsoft DevOps підтримує інтеграцію з різними хмарними сервісами, забезпечуючи гнучкість та масштабованість для застосунків. Це дозволяє використовувати широкий спектр інструментів та сервісів для розробки, тестування та розгортання застосунків [44];

- підтримка та спільнота. Microsoft DevOps має активну спільноту розробників та надає широкий спектр документації, навчальних матеріалів та підтримки. Це дозволяє отримувати відповіді на різні питання, поради та підтримку у разі виникнення проблем [45].

Вибір Microsoft DevOps обґрунтовується його інтегрованою платформою, зручними інструментами для спільної роботи команди, можливістю автоматизації процесів, інтеграцією з хмарними сервісами та наявністю підтримки та активної спільноти. Все це сприяє покращенню ефективності та якості розробки програмного забезпечення.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ПІДТРИМКИ ГОТЕЛЬНОГО СЕРВІСУ

3.1 Функціональні вимоги до проєкту

У системі, що розробляється, повинен бути реалізований наступний функціонал та компоненти [46]:

а) управління бронюваннями:

1) менеджери можуть переглядати, додавати, редагувати та видаляти бронювання для гостей готелю;

2) система надсилає сповіщення про нові бронювання та нагадування про наближення дати заїзду;

б) управління номерами:

1) менеджери можуть переглядати, додавати, редагувати та видаляти інформацію про кімнати в готелі;

2) система надсилає сповіщення про нові бронювання, які вимагають нової кімнати, та нагадування про потребу в ремонті та технічному обслуговуванні кімнат;

в) управління рахунками:

1) менеджери можуть переглядати, додавати, редагувати та видаляти рахунки гостей;

2) система надсилає сповіщення про нові рахунки та нагадування про заборгованість з боку гостей;

г) управління персоналом:

1) менеджери можуть переглядати, додавати, редагувати та видаляти інформацію про співробітників готелю, їхні обов'язки та графіки роботи;

д) управління запасами:

1) менеджери можуть переглядати, додавати, редагувати та видаляти інформацію про товари та послуги, що використовуються в готелі.

Програмний комплекс також повинен бути працездатним на пристрої, що відповідають вимогам:

- операційна система. Програмний комплекс повинен бути сумісним з операційними системами, які підтримуються VisualStudio та Microsoft Azure, такими як Windows, Linux або macOS;
- апаратні вимоги. Програмний комплекс повинен мати вимоги до апаратної частини, які відповідають можливостям та ресурсам пристроїв, на яких він буде використовуватись. Наприклад, процесор, оперативна пам'ять та дисковий простір повинні бути достатніми для виконання програмного комплексу без проблем з продуктивністю;
- мережеве підключення. Оскільки програмний комплекс працює на хмарній платформі, пристрій, на якому він використовується, повинен мати доступ до Інтернету для здійснення з'єднання з хмарним середовищем Microsoft Azure;
- веббраузер. Якщо програмний комплекс має вебінтерфейс, то пристрій повинен мати сумісний веббраузер, який підтримує необхідні технології та стандарти для коректного відображення та взаємодії з програмним комплексом.

3.2 Архітектура системи

Система, що розробляється, має наступну архітектуру та компоненти [47]-[48]:

а) клієнтський інтерфейс:

1) клієнтський інтерфейс є частиною коду, що забезпечує зручний доступ до функціональності системи користувачам;

б) серверна частина:

1) серверна частина системи виконує бізнес-логіку та взаємодіє з базою даних та іншими сервісами;

2) вона реалізована на мові програмування C++ та може використовувати фреймворки, такі як Microsoft Azure SDK або RESTful API;

в) база даних:

1) для зберігання та управління даними системи використовується база даних;

2) можна використовувати ODBC для з'єднання з базою даних та виконання запитів;

3) дані можуть бути збережені в хмарній базі даних, наприклад, Microsoft Azure SQL, або в іншій підтримуваній базі даних;

г) хмарна платформа:

1) для розгортання та керування системою можна використовувати хмарну платформу, наприклад, Microsoft Azure;

2) хмарна платформа забезпечує інфраструктуру для роботи системи, включаючи масштабування, безпеку та доступність;

3) вона також надає можливості інтеграції з іншими сервісами, які можуть бути необхідні для системи готельного сервісу, наприклад, системами бронювання або фінансовими платформами.

Ця архітектура та компоненти дозволяють системі ефективно працювати в обчислювальних системах з будь-якою операційною системою.

Архітектура системи наведена в рисунку 3.1.

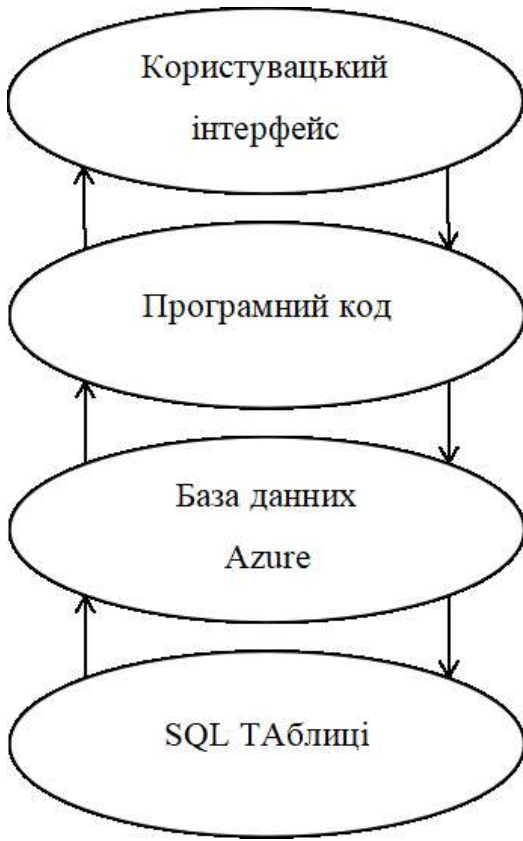


Рисунок 3.1 – Архітектури системи

3.3 Структура бази даних

Для даного застосунку було створено SQLсервер на якому було створено базу даних на хмарному за стосунку MicrosoftAzure. В базу даних було поміщено 6 таблиць.

Таблиці зображені на рисунку 3.2.

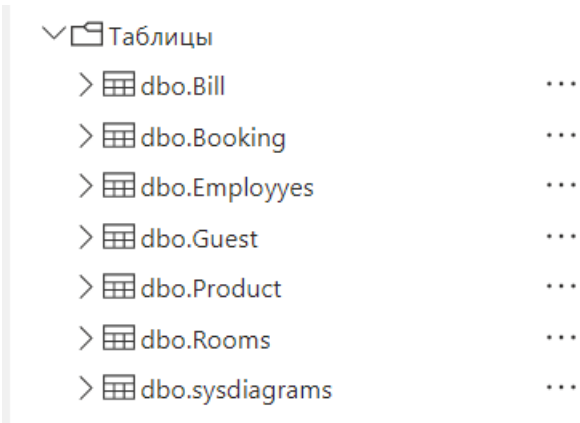


Рисунок 3.2 – Таблиці в базі даних

В кожній таблиці зберігається відповідна інформація:

а) Bill - може зберігати наступні дані:

1) "id" (тип даних: int). Унікальний ідентифікатор рахунку. Це поле встановлено як PRIMARYKEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "Id_guest" (тип даних: int). Ідентифікатор гостя, пов'язаний з таблицею "Guest". Це поле може бути NULL, що означає, що рахунок може не бути пов'язаним з конкретним гостем;

3) "Bill" (тип даних: float). Сума рахунку. Це поле вказує загальну суму, яку гість повинен сплатити.

Ця таблиця "Bill" дозволяє зберігати інформацію про рахунки гостей та пов'язувати їх з конкретними гостями за допомогою зовнішнього ключа "Id_guest";

б) в таблиці "Booking" можуть бути збережені наступні дані:

1) "id" (тип даних: int): Унікальний ідентифікатор бронювання. Це поле встановлено як PRIMARYKEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "id_guest" (тип даних: int): Ідентифікатор гостя, пов'язаний з таблицею "Guest". Це поле може бути NULL, що означає, що бронювання може не бути пов'язаним з конкретним гостем;

3) "ArrivalDate" (тип даних: date): Дата прибуття гостя;

4) "OutDate" (тип даних: date): Дата від'їзду гостя;

5) "room_id" (тип даних: int): Ідентифікатор номера, пов'язаний з таблицею "Rooms". Це поле може бути NULL, що означає, що бронювання може не мати призначеного номера.

Ця таблиця "Booking" дозволяє зберігати інформацію про бронювання гостей, включаючи дати прибуття та від'їзду, пов'язаного гостя;

в) в таблиці "Employuues" можуть бути збережені наступні дані:

1) "id_employee" (тип даних: int). Унікальний ідентифікатор працівника. Це поле встановлено як PRIMARYKEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "name" (типданих: varchar(255)). Ім'я працівника. Це поле може містити текстову інформацію до 255 символів;

3) "position" (тип даних: varchar(255)). Посада працівника. Це поле може містити текстову інформацію до 255 символів;

4) "schedule" (тип даних: varchar(255)). Розклад роботи працівника. Це поле може містити текстову інформацію до 255 символів.

Ця таблиця "Employuues" дозволяє зберігати інформацію про працівників, включаючи їх ім'я, посаду та розклад роботи;

г) в таблиці "Guest" можуть бути збережені наступні дані:

1) "id" (тип даних: int). Унікальний ідентифікатор гостя. Це поле встановлено як PRIMARY KEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "Guest_name" (типданих: varchar(255)). Ім'я гостя. Це поле може містити текстову інформацію до 255 символів.

Ця таблиця "Guest" дозволяє зберігати інформацію про гостей, зокрема їх ім'я. Кожен гість має унікальний ідентифікатор "id", що дозволяє однозначно ідентифікувати кожного гостя в системі;

д) в таблиці "Product" можуть бути збережені наступні дані:

1) "id" (тип даних: int). Унікальний ідентифікатор продукту. Це поле встановлено як PRIMARY KEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "Name_posluga" (тип даних: varchar(255)). Назва продукту або послуги. Це поле може містити текстову інформацію до 255 символів;

3) "Price" (тип даних: float). Ціна продукту або послуги. Це поле зберігає значення ціни у форматі з плаваючою крапкою.

Ця таблиця "Product" дозволяє зберігати інформацію про продукти або послуги. Кожен продукт має унікальний ідентифікатор "id", назву "Name_posluga" і ціну "Price". Це дозволяє вести облік продуктів та їх цін в базі даних;

е) в таблиці "Rooms" можуть бути збережені наступні дані:

1) "id_room" (тип даних: int). Унікальний ідентифікатор номера кімнати. Це поле встановлено як PRIMARYKEY, тому значення будуть унікальними та автоматично збільшуватимуться за допомогою IDENTITY(1,1);

2) "type" (тип даних: varchar(50)). Тип кімнати. Це поле може містити текстову інформацію про тип кімнати, наприклад, "одномісна", "двомісна", "люкс" і т.д.;

3) "capacity" (тип даних: int). Вмістимість кімнати. Це поле зберігає кількість осіб, які можуть поміститись у кімнаті;

4) "available" (тип даних: bit). Прапорець, що вказує на доступність кімнати. Значення "1" означає, що кімната доступна, а значення "0" означає, що кімната недоступна.

Ця таблиця "Rooms" дозволяє зберігати інформацію про кімнати в готелі. Кожна кімната має унікальний ідентифікатор "id_room", тип кімнати "type", вмістимість "capacity" і статус доступності "available". Це дозволяє вести облік кімнат і контролювати їх доступність.

Крім того для всіх таблиць, визначені наступні обмеження:

- PRIMARY KEY CONSTRAINT [PK_Bill]. Це обмеження вказує, що поле "id" є первинним ключем таблиці "Bill". Воно гарантує унікальність значень та прискорює пошук даних;
- FOREIGN KEY CONSTRAINT [FK_Bill_Guest]. Це обмеження встановлює зв'язок між полем "Id_guest" у таблиці "Bill" та полем "id" у таблиці "Guest". Воно забезпечує цілісність даних та вказує, що поле "Id_guest" повинно посилатись на існуючий запис у таблиці "Guest";
- PRIMARY KEY CONSTRAINT [PK_Booking]. Це обмеження вказує, що поле "id" є первинним ключем таблиці "Booking". Воно гарантує унікальність значень та прискорює пошук даних;
- FOREIGN KEY CONSTRAINT [FK_Booking_Guest]. Це обмеження встановлює зв'язок між полем "id_guest" у таблиці "Booking" та полем "id" у таблиці "Guest". Воно забезпечує цілісність даних та вказує, що поле "id_guest" повинно посилатись на існуючий запис у таблиці "Guest";
- FOREIGN KEY CONSTRAINT [FK_Booking_Rooms]. Це обмеження встановлює зв'язок між полем "room_id" у таблиці "Booking" та полем "id_room" у таблиці "Rooms". Воно забезпечує цілісність даних та вказує, що поле "room_id" повинно посилатись на існуючий запис у таблиці "Rooms";
- PRIMARY KEY CONSTRAINT [PK_Employees]. Це обмеження вказує, що поле "id_employee" є первинним ключем таблиці "Employees". Воно гарантує унікальність значень та прискорює пошук даних;
- PRIMARY KEY CONSTRAINT [PK_Guest]. Це обмеження вказує, що поле "id" є первинним ключем таблиці "Guest". Воно гарантує унікальність значень та прискорює пошук даних;
- PRIMARY KEY CONSTRAINT [PK_Product]. Це обмеження вказує, що поле "id" є первинним ключем таблиці "Product". Воно гарантує унікальність значень та прискорює пошук даних;

– PRIMARY KEY CONSTRAINT [PK_Rooms]. Це обмеження вказує, що поле "id_room" є первинним ключем таблиці "Rooms". Воно гарантує унікальність значень та прискорює пошук даних.

Зв'язок таблиць за первинними і вторинними ключами наведено на рисунку 3.3.

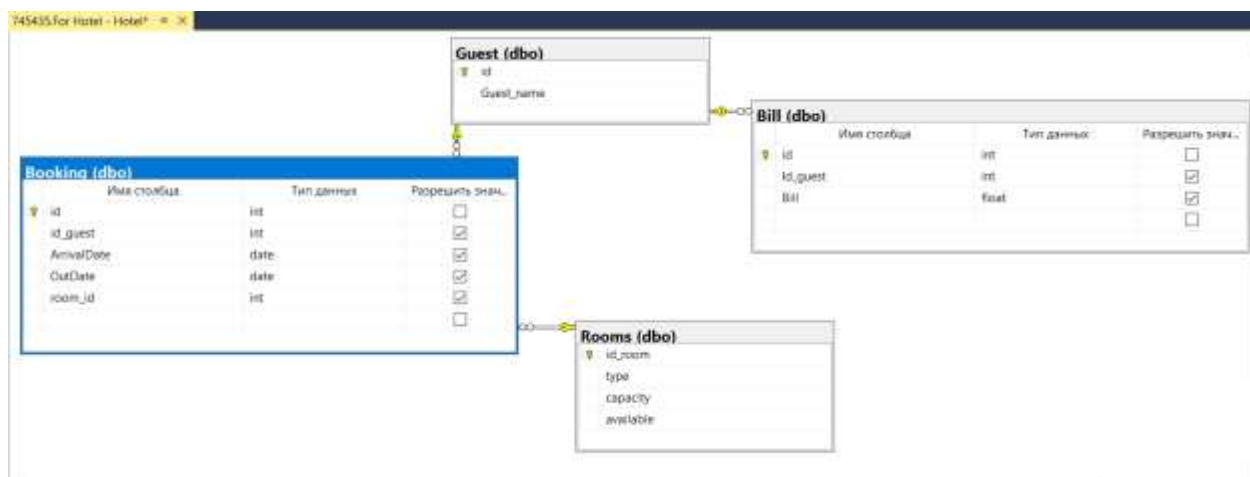


Рисунок 3.3 – Зв'язок таблиць в базі даних

4 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ХМАРНОГО ЗАСТОСУНКУ ДЛЯ ПІДТРИМКИ ГОТЕЛЬНОГО СЕРВІСУ

4.1 Реалізація застосунку мовою C++

Для забезпечення зберігання інформації в базі даних, у головному файлі було виконано підключення до бази даних за допомогою драйвера ODBC [49]. Порівнявши різні драйвери [50], було обрано драйвер ODBC з огляду на його кросплатформеність та підтримку мови програмування C++. Порівняння драйверів наведено в таблиці 4.1.

Таблиця 4.1 – Порівняння драйверів для підключення до бази даних

Драйвер	+	-
ODBC	Кросплатформеність, підтримка C++	Вимагає додаткової конфігурації
JDBC	Підтримка Java	Обмежена підтримка інших мов
ADO.NET	Нативна підтримка .NET	Обмежена кросплатформеність
Node.js драйвер	Підтримка Node.js	Обмежена підтримка інших мов
Python драйвер	Підтримка Python	Обмежена підтримка інших мов
PHP драйвер	Підтримка PHP	Обмежена підтримка інших мов
Ruby драйвер	Підтримка Ruby	Обмежена підтримка інших мов
Go драйвер	Підтримка Go	Обмежена підтримка інших мов

Для включення драйверу до cmakeпроєкту потрібно в файлі CmakeLists.txt додати після TODO:

```
find_package(ODBC REQUIRED)
include_directories(${ODBC_INCLUDE_DIRS})
link_directories(${ODBC_LIBRARY_DIRS})
target_link_libraries(Project_hotel PRIVATE ODBC::ODBC)
```

Реалізація підключення:

```

SetConsoleOutputCP(1251);
SetConsoleCP(1251);
SQLHANDLEhenv;
SQLHANDLEhdbc;
SQLHANDLEhstmt;
SQLRETURNretcode;

    // Ініціалізаціясередовища
    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
    retcode      =      SQLSetEnvAttr(henv,      SQL_ATTR_ODBC_VERSION,
(SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);

    // Створенняпідключення
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
    retcode = SQLDriverConnect(hdbc, NULL, (SQLCHAR*)"Driver={ODBC Driver
18 for SQL Server};Server=tcp:745435.database.windows.net,1433;Database=For
Hotel;Uid=Sgrims;Pwd=Lendra745435;Encrypt=yes;TrustServerCertificate=no;Conne
ction Timeout=30;", SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);

    // Перевіркастатусупідключення
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
        std::cout<<"Підключенняуспішне."<<std::endl;

        // Виконаннязапиту
        retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
        retcode = SQLExecDirect(hstmt, (SQLCHAR*)"SELECT * FROM YourTable",
SQL_NTS);
        update.UpdatesBookings(hdbc);
        system("pause");
        system("cls");
        menu(hdbc);
        // Звільненняресурсів
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        SQLDisconnect(hdbc);
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
        SQLFreeHandle(SQL_HANDLE_ENV, henv);
        exit(0);
    }
    else {
        std::cout<<"Помилкапідключення."<<std::endl;
    }

```

Для виконання кожної функції застосунку було створено окремий заголовок та окреме меню.

Приклади меню та функцій наведені в додатку А.

4.1.1 Управління бронюваннями

Для виконання даної функції в проєкті `make` було створено заголовок `Booking.h`. Заголовок `Booking.h`, представлений в додатку А. В ньому було створено класи `Booking`, що представляють собою окреме бронювання та клас `BookingSystem`.

Клас `BookingSystem` також представлений в Додатку А.

Клас `BookingSystem` виконує операції з управління бронюванням:

- метод `addBooking` додає нове бронювання до системи. Він перетворює рядки з датами в об'єкти типу `std::chrono::system_clock::time_point`, перевіряє правильність дат, перевіряє доступність кімнати та додає бронювання до бази даних. Окрім цього, метод оновлює властивість `available` для кімнати та виводить інформацію про нове бронювання;
- метод `editBooking` змінює існуюче бронювання. Він отримує нові значення для ID гостя, дат заїзду та виїзду, та ID кімнати через `std::cin`. Метод перевіряє валідність дат, оновлює відповідне бронювання в базі даних та виводить оновлену інформацію про бронювання;
- метод `removeBooking` видаляє бронювання з системи. Він оновлює властивість `available` для кімнати, пов'язаної з видаляємим бронюванням, та видаляє бронювання з бази даних;
- метод `parseDate` є допоміжною функцією для розбору рядка з датою.

Клас `Booking` містить приватні поля для збереження інформації про бронювання (ID, ID гостя, дата заїзду, дата виїзду, ID кімнати) та методи доступу до цих полів.

4.1.2 Управління номерами

Для виконання даної функції в проєкті make було створено заголовок Room.h. Заголовок Room.h представлений в додатку А. В ньому було створено класи Room, який представляє окрему кімнату та клас roomSystem, який виконує операції з управління кімнатами.

Аналіз методів класу roomSystem:

- метод addRoom додає нову кімнату до системи. Він зчитує дані про тип кімнати, місткість та наявність з консолі, формує SQL-запит INSERT для вставки даних у таблицю Rooms та виконує його. Якщо вставка пройшла успішно, виводиться повідомлення про успішне додавання, в іншому випадку - про помилку;
- метод displayAllRooms виводить інформацію про всі кімнати з бази даних. Він виконує SQL-запит SELECT для отримання всіх записів з таблиці Rooms, прив'язує стовпці до змінних та виводить отримані значення;
- метод findAvailableRoom знаходить доступну кімнату за заданими критеріями. Він формує SQL-запит SELECT з умовою на тип кімнати, місткість та доступність, виконує запит та виводить знайдену кімнату або повідомлення про її відсутність;
- метод getRoomById отримує інформацію про кімнату за заданим ID. Він формує SQL-запит SELECT з умовою на ID кімнати, виконує запит та виводить отримані значення або повідомлення про відсутність кімнати;
- метод closeRoomAvailability закриває доступність кімнати за заданим ID. Він формує SQL-запит UPDATE для оновлення доступності кімнати та виконує його;
- метод openRoomAvailability відкриває доступність кімнати за заданим ID. Він формує SQL-запит UPDATE для оновлення доступності кімнати та виконує його.

4.1.3 Управління рахунками

Для виконання даної функції в проєктіsmake було створено заголовок BM.h - Заголовок BM.h представлений в додатку А. В якому було створено класи Billякий представляє собою окремий рахунок та клас BillManagement відповідає за управління рахунками (Bill). Аналіз методів класу BillManagement:

- addBill(SQLHDBC dbc, intguestId, doubleamount). Цей метод додає новий рахунок до бази даних. Виконується SQL-запит для вставки нового рахунку в таблицю [dbo].[Bill]. Параметри методу включають з'єднання з базою даних (dbc), ідентифікатор гостя (guestId) та суму рахунку (amount);
- deleteBill(SQLHDBC dbc, intid). Цей метод видаляє рахунок з бази даних за його ідентифікатором. Виконується SQL-запит для видалення рахунку з таблиці [dbo].[Bill]. Параметри методу включають з'єднання з базою даних (dbc) та ідентифікатор рахунку (id);
- editBill(SQLHDBC dbc, intid, doublenewAmount). Цей метод редагує рахунок за його ідентифікатором. Виконується SQL-запит для оновлення суми рахунку в таблиці [dbo].[Bill]. Параметри методу включають з'єднання з базою даних (dbc), ідентифікатор рахунку (id) та нову суму рахунку (newAmount);
- showAllBills(SQLHDBC dbc). Цей метод виводить список всіх рахунків з бази даних. Виконується SQL-запит для вибірки всіх рахунків з таблиці [dbo].[Bill]. Параметр методу - з'єднання з базою даних (dbc);
- sendBillToGuest(SQLHDBC dbc, intbillId). Цей метод відправляє рахунок гостю за його ідентифікатором. Виконується SQL-запит для вибірки рахунку з таблиці [dbo].[Bill] за його ідентифікатором. Якщо рахунок знайдено, він виводиться на екран. Параметри методу включають з'єднання з базою даних (dbc) та ідентифікатор рахунку (billId).

4.1.4 Управління персоналом

Для виконання даної функції в проєкті stake було створено заголовок Manag.h - Заголовок Manag.h представлений в додатку А. В якому було створено клас Employee який представляє собою інформацію про працівника та клас PersonnelManager відповідає за управління персоналом (Employee).

Розглянемо його методи:

- addEmployee(SQLHDBC dbc, conststring&name, conststring&position, conststring&schedule). Цей метод додає нового співробітника до бази даних. Виконується SQL-запит для вставки нового запису в таблицю [dbo].[Employeees]. Параметри методу включають з'єднання з базою даних (dbc), ім'я співробітника (name), посаду (position) та графік роботи (schedule);

- removeEmployee(SQLHDBC dbc, intindex). Цей метод видаляє співробітника з бази даних за його індексом. Виконується SQL-запит для видалення запису з таблиці [dbo].[Employeees] за значенням id_employee, яке відповідає індексу плюс один. Параметри методу включають з'єднання з базою даних (dbc) та індекс співробітника (index);

- editEmployee(SQLHDBC dbc, intid, conststring&name, conststring&position, conststring&schedule). Цей метод редагує інформацію про співробітника за його ідентифікатором (id). Виконується SQL-запит для оновлення запису в таблиці [dbo].[Employeees]. Параметри методу включають з'єднання з базою даних (dbc), ідентифікатор співробітника (id), нове ім'я співробітника (name), нову посаду (position) та новий графік роботи (schedule);

- displayEmployees(SQLHDBC dbc). Цей метод виводить на екран список всіх співробітників з бази даних. Виконується SQL-запит для вибірки всіх записів з таблиці [dbo].[Employeees]. Параметр методу - з'єднання з базою даних (dbc).

4.1.5 Управління запасами

Для виконання даної функції в проєкті `smake` було створено заголовок `Invent.h` - Заголовок `Invent.h` представлений в додатку А. В якому було створено класи `Product` який представляє собою окрему одиницю запасу та клас `Inventory`.

Аналіз методів класу `Inventory`:

- метод `addProduct(SQLHDBC dbc, stringname, doubleprice)`. Цей метод додає новий продукт до бази даних. Він приймає з'єднання з базою даних (`dbc`), назву продукту (`name`) та ціну (`price`). В методі створюється SQL-запит `INSERT`, який вставляє новий рядок з назвою і ціною продукту в таблицю `"Product"`. Запит готується за допомогою функції `SQLPrepare` і виконується за допомогою `SQLExecute`. Успішність виконання запиту перевіряється, і виводиться відповідне повідомлення;

- метод `deleteProduct(SQLHDBC dbc, intid)`. Цей метод видаляє продукт з бази даних за його ідентифікатором (`id`). Він також приймає з'єднання з базою даних (`dbc`). В методі створюється SQL-запит `DELETE`, який видаляє рядок з таблиці `"Product"` за заданим ідентифікатором. Запит готується за допомогою функції `SQLPrepare` і виконується за допомогою `SQLExecute`. Успішність виконання запиту перевіряється, і виводиться відповідне повідомлення;

- метод `editProduct(SQLHDBC dbc, intid, stringname, doubleprice)`. Цей метод редагує продукт у базі даних за його ідентифікатором (`id`). Він приймає з'єднання з базою даних (`dbc`), нову назву продукту (`name`) та нову ціну (`price`). В методі створюється SQL-запит `UPDATE`, який оновлює назву та ціну продукту за заданим ідентифікатором. Запит готується за допомогою функції `SQLPrepare` і виконується за допомогою `SQLExecute`. Успішність виконання запиту перевіряється, і виводиться відповідне повідомлення;

- метод `showAllProducts(SQLHDBC dbc)`. Цей метод виводить всі продукти з бази даних. Він приймає з'єднання з базою даних (`dbc`). У методі

створюється SQL-запит SELECT, який вибирає всі рядки з таблиці "Product". Запит виконується за допомогою функції SQLExecDirect. Результати запиту отримуються за допомогою функції SQLGetData, і дані виводяться на екран.

4.2 Реалізація хмарного кросплатформеного застосунку

Даний застосунок був розроблений у проєкті CMake з використанням Visual Studio, що дозволяє його використання на операційних системах Linux, Windows і macOS [51]. Для цього необхідно встановити відповідне програмне забезпечення та драйвери.

Для реалізації хмарного зберігання було обрано платформу Microsoft DevOps [52]. За допомогою функціоналу Visual Studio Git був створений репозиторій, який разом з усіма файлами був завантажений до Microsoft DevOps за наступним посиланням: <https://dev.azure.com/> [53].

Файли проєкту в репозиторію зображені на рисунку 4.1.

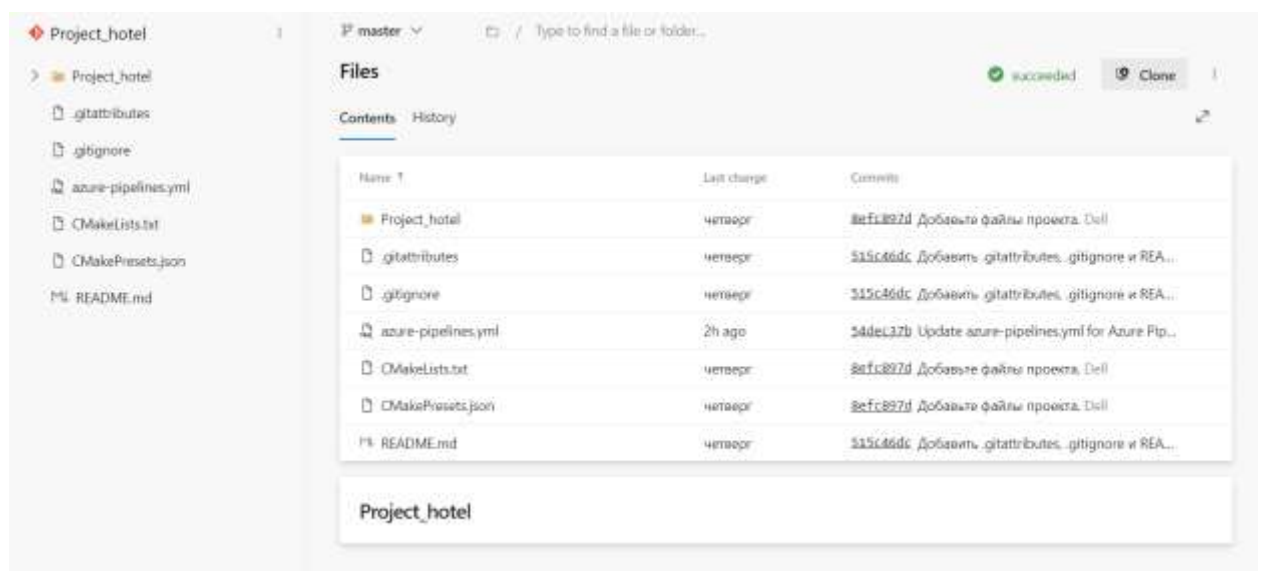


Рисунок 4.1 – Репозиторій Project_hotel на MicrosoftDevops

В MicrosoftDevops за допомогою функції pipelines було виконано збірку та перевірено саму програму на помилки.

Збір проєкту зображено на рисунку 4.2.

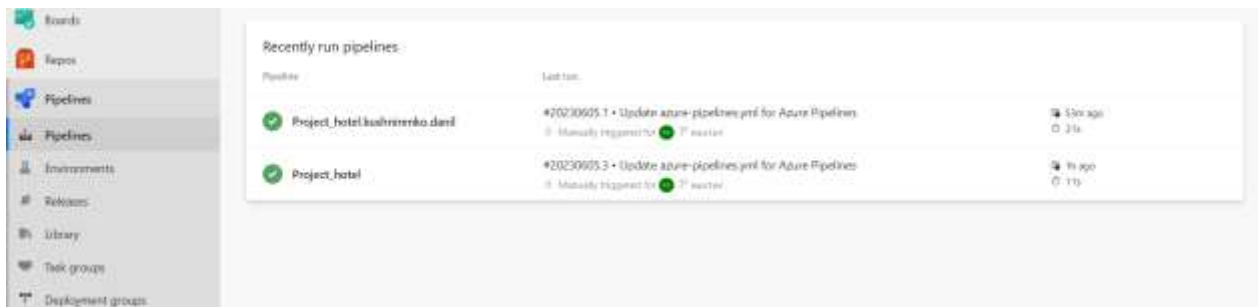


Рисунок 4.2 – Збір проєкту на MicrosoftDevops

4.3 Особливості використання застосунку

Для початку роботи зі застосунком потрібно завантажити код з хмарного репозиторію MicrosoftDevops, для цього потрібно перейти на сайт за посиланням Project_hotel - Repos (azure.com) та ввести логін і пароль організації яка буде володіти готелем.

Клонування коду зображено на рисунку 4.3.

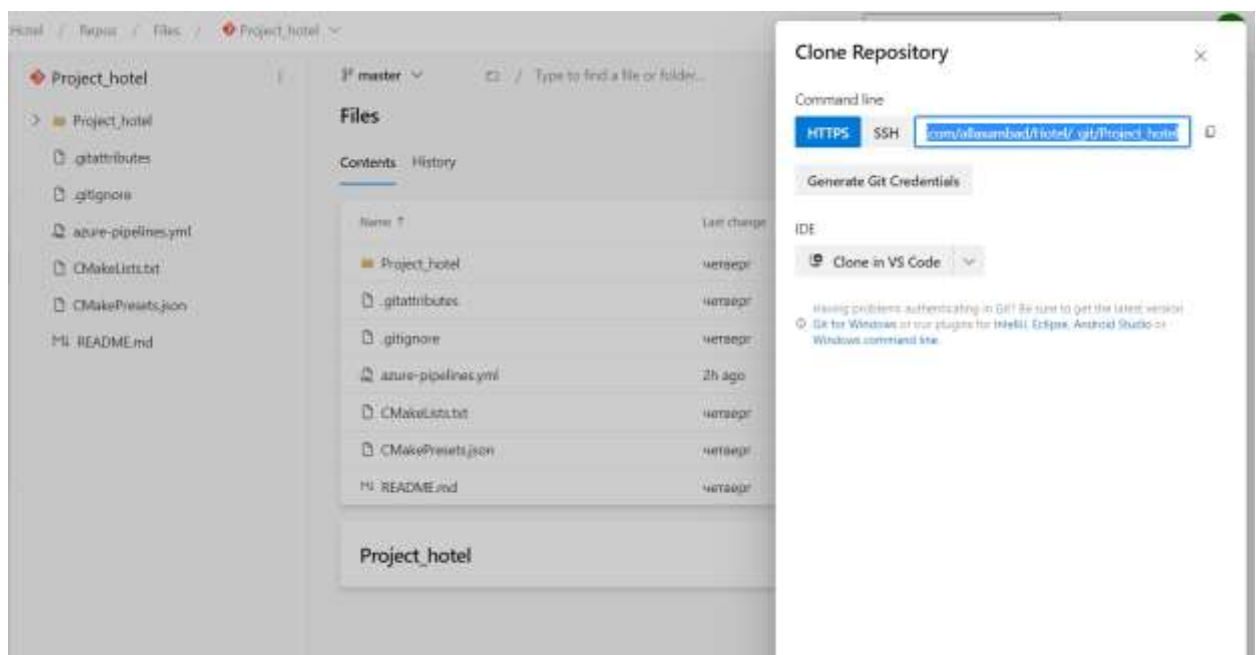


Рисунок 4.3 – Меню клонування коду

Наступним етапом буде завантаження драйвера ODBC [23] для підключення до бази даних які знаходяться на хмарному сервісі MicrosoftAzure. Завантажити цей драйвер можливо за посиланням Download

ODBC Driverfor SQL Server - ODBC Driverfor SQL Server | Microsoft Learn. Після завантаження драйверу потрібно перейти до сайту Microsoft Azure - Главная - Microsoft Azure, де теж потрібно авторизуватись, та занести ір адресу в правила брандмауера для надання доступу до бази даних яка, знаходиться на цьому сайті.

4.4 Особливості взаємодії з користувачем

Даний застосунок використовує SQL запити для взаємодії з базою даних, яка містить SQL таблиці, і здійснює зміни в цих таблицях. Після запуску програмного коду відбувається підключення до бази даних Microsoft Azure за допомогою ODBC драйвера через рядок підключення.

Для здійснення підключення до бази даних Microsoft Azure за допомогою ODBC драйвера використовується рядок підключення, який містить необхідні параметри для з'єднання з базою даних [54]. Цей рядок підключення передається функції SQLDriverConnect, яка встановлює з'єднання з базою даних [55]. Одним з популярних ODBC драйверів для підключення до бази даних Microsoft Azure є "ODBC Driver 18 for SQL Server" [56].

Рядок підключення зображений на рисунку 4.4.

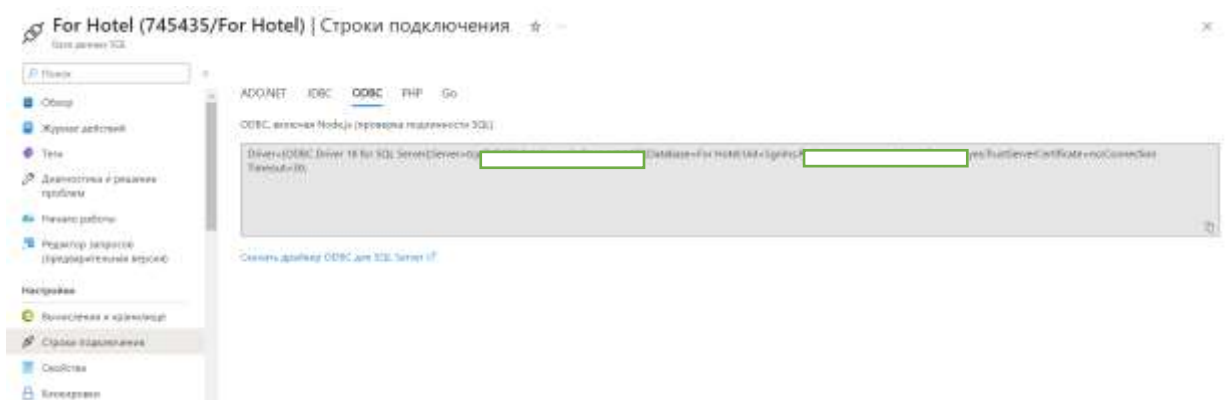


Рисунок 4.4 – Рядок підключення бази даних

Після підключення в консоль виводиться меню з яким може взаємодіяти користувач.

Меню зображене на рисунку 4.5.

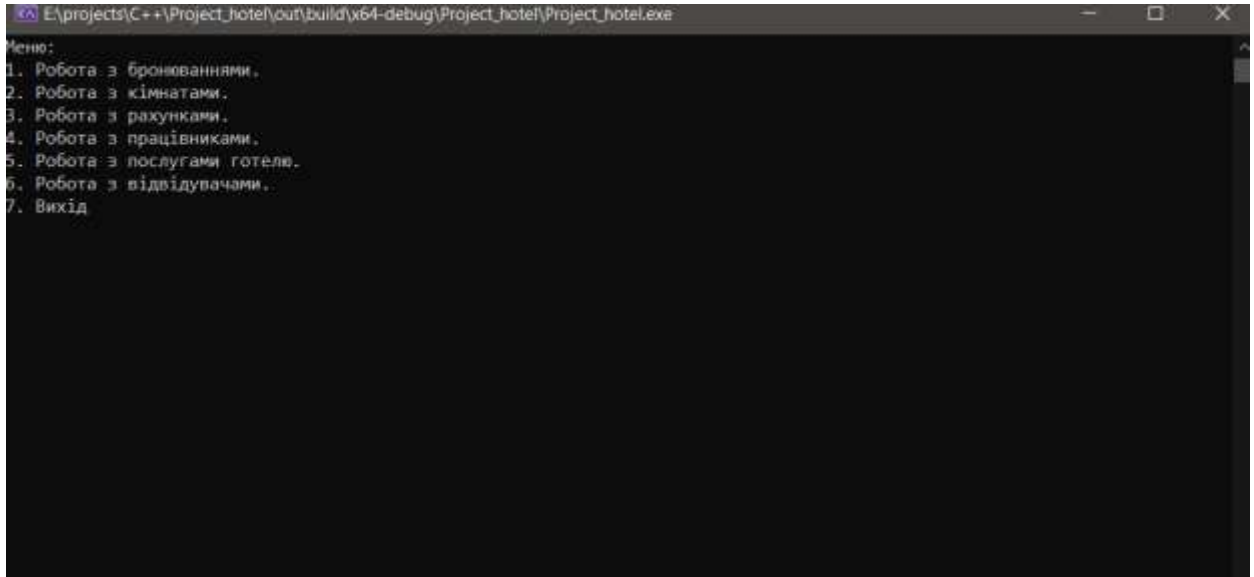


Рисунок 4.5 – Меню

Ввівши номер функції з якою хоче взаємодіяти користувач та нажавши клавішу Enter в консолі зміниться меню на інше де будуть описані інші функції.

Підменю для функції робота з бронюваннями зображене на рисунку 4.6.

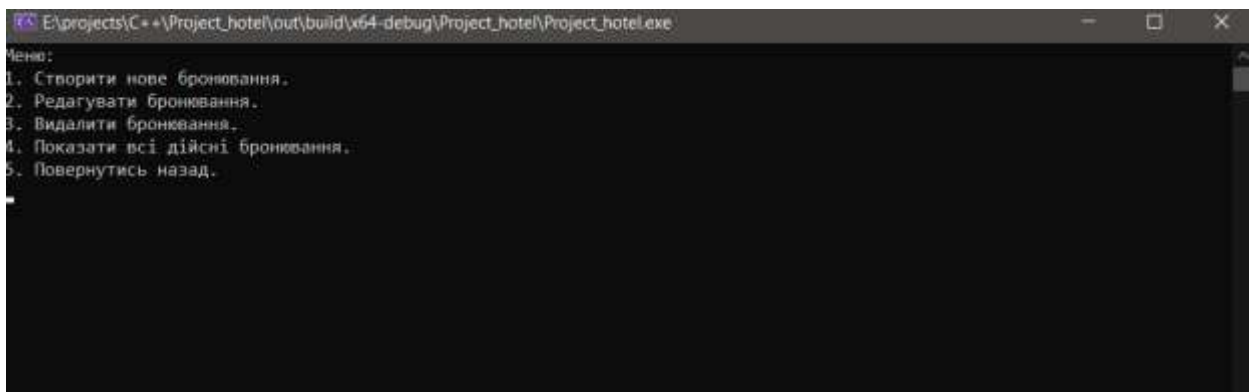


Рисунок 4.6 – Меню функції Робота з бронюваннями

Далі користувач вводить наступний номер функції яку він хоче виконати. Після чого в Консолі з'являться рядки інформації про відвідувача та готель які потрібно заповнити користувачу. Після їх заповнення програма виконує деякі перевірки й надсилає запит до відповідної SQLтаблиці та редагує данні, якщо все пройшло успішно з'явиться нове повідомлення наприклад про нове бронювання.

Повідомлення про створене бронювання зображене на рисунку 4.7.

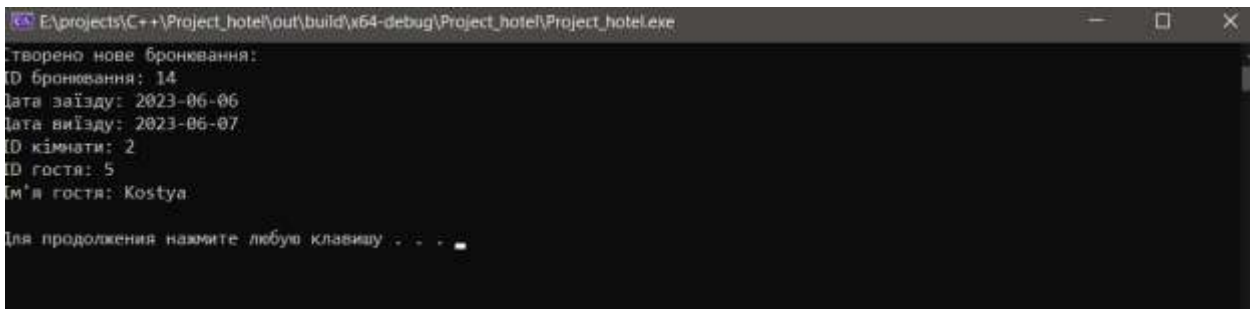


Рисунок 4.7 – Повідомлення про створення бронювання

Після чого користувач натиснувши на будь-яку клавішу повернутись до попереднього меню і виконати наступну функцію або вийти з програми.

4.5 Перевірки реалізовані під час виконання коду

Перед посиланням та виконанням SQLзапитів було реалізовано декілька перевірок:

- перевірка дати;
- перевірка кімнати на доступність (вільна, зайнята);
- перевірка надходження та виконання SQL запиту.

Якщо програма не пройшла перевірку то будуть з'являтися відповідні повідомлення.

Повідомлення про перевірки зображено на рисунках 4.8-4.9.

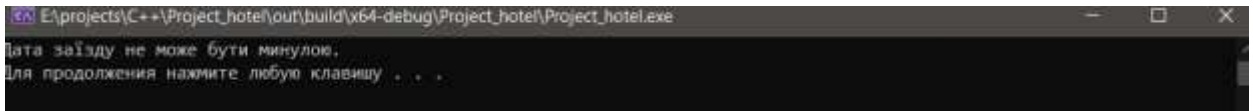


Рисунок 4.8 – Повідомлення перевірок

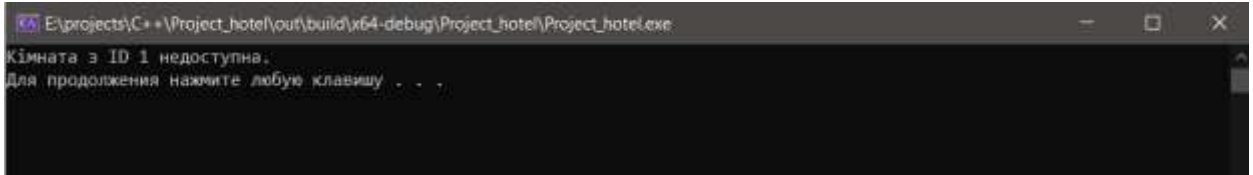


Рисунок 4.9 – Повідомлення перевірок

4.6 Відлагодження та тестування розробленого сервісу, порівняння результатів роботи з існуючими аналогами

Після компіляції застосунку для підтримки готельного сервісу було проведено налагодження програмного коду та наступне тестування даного макету. Під час тестування були виявлені наступні недоліки: відсутність графічного інтерфейсу та неможливість зберігати дані на українській мові за допомогою SQL таблиць.

Це обґрунтовано тим, що безкоштовна версія Microsoft Azure не дозволяє змінювати кодування SQL таблиць, а сама програма написана на C++, що не передбачає графічних дизайнів. Під час налагодження був виявлений недолік у підключенні до баз даних, який був виправлений шляхом оновлення драйверу ODBC.

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було проведено аналіз найбільш відомих і популярних на сьогодні систем управління готельним бізнесом. Були розглянуті основні переваги використання Microsoft Azure, особливості розробки в середовищі Visual Studio, а також вивчені різні підходи до підключення до бази даних Microsoft Azure. Крім того, весь проєкт було внесено до платформи Microsoft DevOps.

У роботі були сформульовані функціональні вимоги до програмного продукту, розроблена архітектура системи та створений базовий варіант (макет) застосунку для підтримки готельного сервісу.

Розроблений застосунок дозволяє користувачам спрощувати процеси бронювання кімнат, управління працівниками, керування рахунками відвідувачів та управління запасами готелю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Hotelogix - Official website of Hotelogix, a gloomy solution for hotel management [Electronic resource]. – Access mode: <https://www.hotelogix.com/>.
2. OperaPMS - Official website of OracleOperaPropertyManagementSystem (PMS), a local hotel management solution [Electronic resource]. – Access mode: <https://www.oracle.com/hospitality/hotel-property-management/hotel-pms-software/>.
3. Cloudbeds - The official website of Cloudbeds, a gloomy solution for hotel management [Electronic resource]. – Access mode: <https://www.cloudbeds.com/>.
4. Hotello - Official website of Hotello, a gloomy solution for hotel management [Electronic resource]. – Access mode: <https://hotello.com/>.
5. "Cloud-Based Hotel Management Software: The Future of Hotel Technology." Hotel Tech Report [Electronic resource]. – Access mode: <https://hoteltechreport.com/>.
6. Wiggins, David "How Cloud Computing Helps Meet Compliance Requirements" Microsoft Azure Blog [Electronic resource] / David Wiggins. – 2021. – Access mode: <https://azure.microsoft.com/en-us/blog/>.
7. Pizzi, Michael "Hotel Technology: How to Choose the Right Property Management System" Hotel Tech Report [Electronic resource] / Michael Pizzi. – 2018. – Access mode: <https://hoteltechreport.com/>.
8. Hirsch, Michael "How to Choose Hotel Management Software" Oracle [Electronic resource] / Michael Hirsch. – 2023. – Access mode: <https://www.oracle.com/hospitality/products/hotel-channel-management-software/>.
9. FrontdeskAnywhere. "Hotel Property Management System." FrontdeskAnywhere [Electronic resource]. – Access mode: <https://www.frontdeskanywhere.com/>.

10. Hotelogix. "Cloud PMS." Hotelogix [Electronic resource]. – Access mode: <https://www.hotelogix.com/>.
11. Oracle. "Oracle Hospitality Hotel Property Management System." Oracle [Electronic resource]. – Access mode: <https://www.oracle.com/hospitality/hotel-property-management/hotel-pms-software/>.
12. Oracle Hospitality. "Oracle Hospitality" [Electronic resource]. – Access mode: www.oracle.com/industries/hospitality/.
13. Booking.com. "Знайдіть помешкання для наступної подорожі". [Електрон. ресурс]. – Режим доступу: www.booking.com/index.uk.html.
14. Cloudbeds. "Hotel Management Software" Cloudbeds [Electronic resource]. – Access mode: www.cloudbeds.com/.
15. RoomKeyPMS. "Hotel Property Management System." RoomKeyPMS [Electronic resource]. – Access mode: www.roomkeypms.com/.
16. Amazon Web Services (AWS). Official website [Electronic resource]. – Access mode: aws.amazon.com/.
17. Microsoft Azure. Official website [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-gb/>.
18. Google Cloud Platform (GCP). Official website [Electronic resource]. – Access mode: cloud.google.com/.
19. RightScale. "2019 State of the Cloud Report" Flexera [Electronic resource]. – Access mode: <https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf>.
20. "Amazon AWS vs. Microsoft Azure: What's the Difference?" [Electronic resource]. – Access mode: <https://www.simplilearn.com/tutorials/cloud-computing-tutorial/aws-vs-azure>.
21. MicrosoftAzure. "Cloud computing" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-cloud-computing/>.

22. AWS. "Infrastructure AWS" [Electronic resource]. – Access mode: <https://aws.amazon.com/what-is-cloud-computing/>.
23. GoogleCloud. "What are cloud services?" [Electronic resource]. – Access mode: <https://cloud.google.com/what-is-cloud-computing>.
24. Netflix TechBlog. "Cloud Computing at Netflix" [Electronic resource]. – Access mode: <https://netflixtechblog.com/tagged/cloud-computing>.
25. IBMCloud. "What are cloud services?" [Electronic resource]. – Access mode: <https://www.ibm.com/cloud/learn/what-are-cloud-services>.
26. MicrosoftAzure. "Overview of Azure services" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-gb/products/>.
27. Microsoft Azure. "Azure Virtual Machines" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/virtual-machines/>.
28. Microsoft Azure. "Azure Cloud Services" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/cloud-services/>.
29. Microsoft Azure. "Azure Service Fabric" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/service-fabric/>.
30. Microsoft Azure. "Azure Functions" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/functions/>.
31. Microsoft Azure. "Azure Content Delivery Network (CDN)" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/cdn/>.
32. Microsoft Azure. "Azure ExpressRoute" – [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/expressroute/>.
33. Microsoft Azure. "Azure Virtual Network" – [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/virtual-network/>.
34. Microsoft Azure. "Azure DNS" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/dns/>.
35. Microsoft Azure. "Azure Disk Storage" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/storage/disks/>.

36. Microsoft Azure. "Azure Blob Storage" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/storage/blobs/>.
37. Microsoft Azure. "Azure File Storage" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/storage/files/>
38. Microsoft Azure. "Azure Queue Storage" [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/services/storage/queues/>.
39. Prasad, S. "Advantages and Disadvantages of C++ Language," / S. Prasad // International Journal of Research and Innovation in Computer Science. – 2018. – Vol. 5, № 6, – P. 114-117.
40. Microsoft Docs, "What is Azure DevOps?," Microsoft, [Online] [Electronic resource]. – Access mode: <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-azure-devops>.
41. Gangadharappa, S. R. "Enhancing Agility in Software Development Using Azure DevOps" / S. R. Gangadharappa, D. B. Phaneendra // International Journal of Computer Science and Mobile Computing. – 2020. – Vol. 9, № 3 – P. 119-124.
42. Kar, P. B. "Enhancing Agile Development using Microsoft Azure DevOps" / P. B. Kar, K. M. Naik // International Journal of Computer Science and Information Security. – 2019. – Vol. 17, №. 3 – P. 102-107.
43. Al-Hajjaji, P. "Implementation of a CI/CD Pipeline for Infrastructure as Code using Azure DevOps and Terraform" / P. Al-Hajjaji, A. Bauer // In Proceedings of the International Conference on Advanced Information Systems Engineering. – 2020. – P. 64-72.
44. Maity, B. "Continuous Deployment of Microservices-Based Applications with Azure DevOps" / B. Maity, M. N. Gupta // In Proceedings of the International Conference on Advances in Computing, Communications and Informatics. – 2020. – P. 545-551.
45. Schrader, A. "Developing Quality Software using Azure DevOps: A Real Options Perspective" / A. Schrader, N. H. Madhavji // In Proceedings of the

International Conference on Evaluation of Novel Approaches to Software Engineering. – 2021. – P. 372-382.

46. Patel "Design and Implementation of Hotel Management System" / Patel, S. Gupta // International Journal of Scientific and Research Publications. – 2013. – Vol. 3, № 6 – P. 1-6.

47. Smith, J. "Design and Implementation of a Hotel Management System" / J. Smith // Proceedings of the International Conference on Computer Science and Software Engineering. – 2015. – Vol. 2 – P. 123-128.

48. Johnson, K. "Hotel Management System Architecture" / K. Johnson // International Journal of Computer Applications. – 2016. – Vol. 150, № 6, P. 12-15.

49. Microsoft Docs, "ODBC Driver" - Microsoft's documentation on the ODBC driver – [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server?view=sql-server-ver16>.

50. Anderson, S. "Comparison of Database Drivers" - which provides a comparison of different drivers for working with databases [Electronic resource] / S. Anderson. – 2022. – Access mode: <https://www.ibm.com/docs/en/db2-for-zos/12>.

51. CMake – official website CMake [Electronic resource]. – Access mode: <https://cmake.org/>.

52. MicrosoftDevOps – official website MicrosoftDevOps [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/products/devops/>.

53. AzureDevOps – official website AzureDevOps (formerly known as MicrosoftDevOps) – official website MicrosoftDevOps [Electronic resource]. – Access mode: <https://azure.microsoft.com/en-us/products/devops/>.

54. MicrosoftAzure database connection string - Microsoft documentation that provides a description of the parameters used to connect to a MicrosoftAzure database through an ODBC driver [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server?view=sql-server-ver16/>.

55. SQLDriverConnect - Microsoft documentation that explains how to use the SQLDriverConnect function to connect to a database using an ODBC driver [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/sql/odbc/reference/syntax/sqldriverconnect-function?view=sql-server-ver15>.

56. ODBC Driver 18 for SQL Server – Official Microsoft document describing "ODBC Driver 18 for SQL Server", which is one of the available ODBC drivers for connecting to Microsoft Azure Database [Electronic resource]. – Access mode: <https://learn.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server?view=sql-server-ver15>.

ДОДАТОК А
Текст програми

A.1 Текст файла HeaderBM.h

```
#pragma once
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <iterator>
#include <sql.h>
#include <sqlext.h>

// Класрахунку
class Bill {
public:
    Bill(int id, int guestId, double amount) {
        this->id = id;
        this->guestId = guestId;
        this->amount = amount;
    }

    int getId() const {
        return id;
    }

    int getGuestId() const {
        return guestId;
    }

    double getAmount() const {
        return amount;
    }

    void setAmount(double amount) {
        this->amount = amount;
    }

private:
    int id;
    int guestId;
    double amount;
};
```

```

// Класуправліннярахунками
class BillManagement {
private:
    std::vector<Bill> bills;

public:
    // Додатиновийрахунок
    void addBill(SQLHDBC dbc, int guestId, double amount) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        // Формуваннязапитудлядодаванняновогорахунку
        std::string query = "INSERT INTO [dbo].[Bill] (Id_guest, Bill) VALUES
(" + std::to_string(guestId) + ", " + std::to_string(amount) + ")";

        ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            std::cout<<"Рахунок успішно додано.\n";
        }
        else {
            std::cout<<"Помилка при виконанні запиту.\n";
        }

        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

    // Видалити рахунок за ідентифікатором
    void deleteBill(SQLHDBC dbc, int id) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        // Формуваннязапиту для видалення рахунку
        std::string query = "DELETE FROM [dbo].[Bill] WHERE id = " +
std::to_string(id);

        ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            std::cout<<"Рахунок успішно видалено.\n";
        }
    }
}

```

```

else {
    std::cout<<"Помилка при виконанні запиту.\n";
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// Редагувати рахунок за ідентифікатором
void editBill(SQLHDBC dbc, int id, double newAmount) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту для редагування рахунку
    std::string query = "UPDATE [dbo].[Bill] SET Bill = " +
    std::to_string(newAmount) + " WHERE id = " + std::to_string(id);

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        std::cout<<"Рахунок успішно відредаговано.\n";
    }
    else {
        std::cout<<"Помилка при виконанні запиту.\n";
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void showAllBills(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту для вибірки всіх рахунків
    std::string query = "SELECT * FROM [dbo].[Bill]";

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        std::cout<<"Список рахунків:\n";

        // Отримання результатів запиту
        while (SQLFetch(stmt) == SQL_SUCCESS) {
            SQLINTEGER id;

```

```

        SQLINTEGER guestId;
        SQLDOUBLE amount;

        // Отримання значень колонок
        SQLGetData(stmt, 1, SQL_INTEGER, &id, sizeof(id), NULL);
        SQLGetData(stmt, 2, SQL_INTEGER, &guestId, sizeof(guestId), NULL);
        SQLGetData(stmt, 3, SQL_DOUBLE, &amount, sizeof(amount), NULL);

        // Виведення значень рахунку
        std::cout << "ID: " << id << ", Guest ID: " << guestId << ", Amount: " <<
amount << "\n";
    }
    else {
        std::cout << "Помилка при виконанні запиту.\n";
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void sendBillToGuest(SQLHDBC dbc, int billId) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту для отримання рахунку за його ідентифікатором
    std::string query = "SELECT * FROM [dbo].[Bill] WHERE id = " +
std::to_string(billId);

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        // Перевірка, чи є результат запиту
        if (SQLFetch(stmt) == SQL_SUCCESS) {
            SQLINTEGER id;
            SQLINTEGER guestId;
            SQLDOUBLE amount;

            // Отримання значень рахунку
            SQLGetData(stmt, 1, SQL_INTEGER, &id, sizeof(id), NULL);
            SQLGetData(stmt, 2, SQL_INTEGER, &guestId, sizeof(guestId), NULL);
            SQLGetData(stmt, 3, SQL_DOUBLE, &amount, sizeof(amount), NULL);

```

```

// Відправлення рахунку гостю (в цьому прикладі просто виводимо його на
екран)

std::cout<<"Рахунок для гостя з ID"<<guestId<<":\n";
std::cout<<"ID: "<< id << ", Amount: "<< amount << "\n";
    }
    else {
std::cout<<"Рахунок з ID "<<billId<<" не знайдено.\n";
    }
}
else {
std::cout<<"Помилка при виконанні запиту.\n";
}
SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}
};

```

A.2 Текст файлу Header Booking.h

```

#pragma once
#pragma once
#include <iostream>
#include <chrono>
#include <sql.h>
#include <sqlext.h>
#include <string>

class Booking {
public:
    Booking(int id, int guestId, std::string arrivalDate, std::string
checkoutDate, int roomId) {
        this->id = id;
        this->guestId = guestId;
        this->arrivalDate = arrivalDate;
        this->checkoutDate = checkoutDate;
        this->roomId = roomId;
    }

    int getId() const {
        return id;
    }
}

```

```

        int getGuestId() const {
            return guestId;
        }

        std::string getArrivalDate() const {
            return arrivalDate;
        }

        std::string getCheckoutDate() const {
            return checkoutDate;
        }

        int getRoomId() const {
            return roomId;
        }

        ~Booking() {}

private:
    int id;
    int guestId;
    std::string arrivalDate;
    std::string checkoutDate;
    int roomId;
};

// Клас, що представляє систему управління бронюванням
class BookingSystem {
public:
    void addBooking(std::string arrivalDate, std::string checkoutDate,
int roomId, int guestId, SQLHDBC dbc) {
        // Перетворення рядків з датами в
об'єкти типу std::chrono::system_clock::time_point
        std::chrono::system_clock::time_point arrivalTime, checkoutTime;

        // Перетворення рядка з датою заїзду в
об'єкти типу std::chrono::system_clock::time_point
        {
            std::istringstream iss(arrivalDate);
            std::tm tm = {};
            iss >> std::get_time(&tm, "%Y-%m-%d");
            arrivalTime =
std::chrono::system_clock::from_time_t(std::mktime(&tm));

```

```

    }

    // Перетворення рядка з датою виїзду в
    об'єкт типу std::chrono::system_clock::time_point
    {
        std::stringstream iss(checkoutDate);
        std::tm tm = {};
        iss >> std::get_time(&tm, "%Y-%m-%d");
        checkoutTime =
std::chrono::system_clock::from_time_t(std::mktime(&tm));
    }

    // Перевірка на правильність дати
    std::chrono::system_clock::time_point currentTime =
std::chrono::system_clock::now();

    if (arrivalTime < currentTime) {
std::cout << "Дата заїзду не може бути минулою." << std::endl;
return;
    }

    if (checkoutTime <= arrivalTime) {
std::cout << "Дата виїзду повинна бути після дати заїзду." << std::endl;
return;
    }

    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Перевірка кімнати на доступність
    SQLHSTMT checkRoomAvailabilityStmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc,
&checkRoomAvailabilityStmt);
    ret = SQLPrepare(checkRoomAvailabilityStmt, (SQLCHAR*)"SELECT
[available] FROM [dbo].[Rooms] WHERE [id_room] = ?", SQL_NTS);
    ret = SQLBindParameter(checkRoomAvailabilityStmt, 1,
SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &roomId, 0, NULL);
    ret = SQLExecute(checkRoomAvailabilityStmt);

    SQLCHAR outAvailable[256];
    SQLLEN availableLen;
    ret = SQLFetch(checkRoomAvailabilityStmt);

```

```

        if (ret == SQL_SUCCESS) {
            ret = SQLGetData(checkRoomAvailabilityStmt, 1, SQL_C_CHAR,
outAvailable, sizeof(outAvailable), &availableLen);
            if (ret == SQL_SUCCESS) {
                bool available = (outAvailable[0] == '1');
                if (!available) {
std::cout<<"Кімната з ID "<<roomId<<" недоступна."<< std::endl;
SQLFreeHandle(SQL_HANDLE_STMT, checkRoomAvailabilityStmt);
return;
                }
            }
        }

SQLFreeHandle(SQL_HANDLE_STMT, checkRoomAvailabilityStmt);

// Додавання бронювання
SQLHSTMT insertStmt;
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &insertStmt);
ret = SQLPrepare(insertStmt, (SQLCHAR*)"INSERT INTO
[dbo].[Booking] ([ArrivalDate], [OutDate], [room_id], [id_guest]) VALUES (?,
?, ?, ?)", SQL_NTS);

ret = SQLBindParameter(insertStmt, 1, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_TYPE_DATE, 0, 0, (SQLPOINTER)arrivalDate.c_str(), 0, NULL);
ret = SQLBindParameter(insertStmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_TYPE_DATE, 0, 0, (SQLPOINTER)checkoutDate.c_str(), 0, NULL);
ret = SQLBindParameter(insertStmt, 3, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER, 0, 0, &roomId, 0, NULL);
ret = SQLBindParameter(insertStmt, 4, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER, 0, 0, &guestId, 0, NULL);

ret = SQLExecute(insertStmt);

if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
std::cout<<"Не вдалося додати бронювання."<<std::endl;
}
else {
// Отримання id нового бронювання
SQLHSTMT selectBookingIdStmt;
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &selectBookingIdStmt);
ret = SQLExecDirect(selectBookingIdStmt, (SQLCHAR*)"SELECT
@@IDENTITY", SQL_NTS);

```

```

        SQLCHAR outBookingId[256];
        SQLLEN bookingIdLen;
        ret = SQLFetch(selectBookingIdStmt);
        if (ret == SQL_SUCCESS) {
            ret = SQLGetData(selectBookingIdStmt, 1, SQL_C_CHAR,
outBookingId, sizeof(outBookingId), &bookingIdLen);
            if (ret == SQL_SUCCESS) {
                int bookingId =
std::atoi(reinterpret_cast<char*>(outBookingId));

                // Отримання імені гостя
                SQLHSTMT selectGuestNameStmt;
                ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc,
&selectGuestNameStmt);
                ret = SQLPrepare(selectGuestNameStmt,
(SQLCHAR*)"SELECT [Guest_name] FROM [dbo].[Guest] WHERE [id] = ?", SQL_NTS);
                ret = SQLBindParameter(selectGuestNameStmt, 1,
SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &guestId, 0, NULL);
                ret = SQLExecute(selectGuestNameStmt);

                SQLCHAR outGuestName[256];
                SQLLEN guestNameLen;
                ret = SQLFetch(selectGuestNameStmt);
                if (ret == SQL_SUCCESS) {
                    ret = SQLGetData(selectGuestNameStmt, 1,
SQL_C_CHAR, outGuestName, sizeof(outGuestName), &guestNameLen);
                    if (ret == SQL_SUCCESS) {
                        std::string guestName(reinterpret_cast<char*>(outGuestName));

                        std::cout<<"Створено нове бронювання:\n"
<<"ID бронювання: "<<bookingId<<"\n"
<<"Дата заїзду: "<<arrivalDate<<"\n"
<<"Дата виїзду: "<<checkoutDate<<"\n"
<<"ID кімнати: "<<roomId<<"\n"
<<"ID гостя: "<<guestId<<"\n"
<<"Ім'я гостя: "<<guestName<<"\n"

                        <<std::endl;

                        // Оновлення властивості available
для кімнати
                        SQLHSTMT updateRoomAvailabilityStmt;
                        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc,
&updateRoomAvailabilityStmt);

```

```

                                ret                                =
SQLPrepare(updateRoomAvailabilityStmt, (SQLCHAR*)"UPDATE [dbo].[Rooms] SET
[available] = 0 WHERE [id_room] = ?", SQL_NTS);
                                ret                                =
SQLBindParameter(updateRoomAvailabilityStmt, 1, SQL_PARAM_INPUT, SQL_C_SLONG,
SQL_INTEGER, 0, 0, &roomId, 0, NULL);
                                ret                                =
SQLExecute(updateRoomAvailabilityStmt);

    SQLFreeHandle(SQL_HANDLE_STMT, updateRoomAvailabilityStmt);
    }
}

SQLFreeHandle(SQL_HANDLE_STMT, selectGuestNameStmt);
    }
}

SQLFreeHandle(SQL_HANDLE_STMT, selectBookingIdStmt);
    }

SQLFreeHandle(SQL_HANDLE_STMT, insertStmt);
    }

void editBooking(int id, SQLHDBC dbc)
{
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    ret = SQLPrepare(stmt, (SQLCHAR*)"UPDATE [dbo].[Booking] SET
[id_guest] = ?, [ArrivalDate] = ?, [OutDate] = ?, [room_id] = ? WHERE [id] =
?", SQL_NTS);

    int guestId;
    std::string arrivalDateStr, checkoutDateStr;
    int roomId;

    // Введення нових значень через cin
    std::cout<<"Введіть ID гостя: ";
    std::cin>>guestId;
    std::cout<<"Введіть дату заїзду (рік-місяць-день): ";
    std::cin>>arrivalDateStr;
    std::cout<<"Введіть дату виїзду (рік-місяць-день): ";
    std::cin>>checkoutDateStr;
    std::cout<<"Введіть ID кімнати: ";

```

```

std::cin>>roomId;

std::chrono::system_clock::time_point now =
std::chrono::system_clock::now();
std::time_t nowTime = std::chrono::system_clock::to_time_t(now);
std::tm localTime = {};
localtime_s(&localTime, &nowTime);
int currentYear = localTime.tm_year + 1900;
int currentMonth = localTime.tm_mon + 1;
int currentDay = localTime.tm_mday;

std::string currentDateStr = std::to_string(currentYear) + "-" +
std::to_string(currentMonth) + "-" + std::to_string(currentDay);

std::chrono::system_clock::time_point arrivalDateTP =
std::chrono::system_clock::from_time_t(std::mktime(parseDate(arrivalDateStr)
));
std::chrono::system_clock::time_point checkoutDateTP =
std::chrono::system_clock::from_time_t(std::mktime(parseDate(checkoutDateStr)
));

if (arrivalDateTP < now || checkoutDateTP < now ||
arrivalDateTP > checkoutDateTP) {
std::cout<<"Введено недійсні дати."<< std::endl;
return;
}

std::cout<<"Введено діючі дати."<< std::endl;

ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, &guestId, 0, NULL);
ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_TYPE_DATE, 0, 0, (SQLPOINTER)arrivalDateStr.c_str(), 0, NULL);
ret = SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_TYPE_DATE, 0, 0, (SQLPOINTER)checkoutDateStr.c_str(), 0, NULL);
ret = SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, &roomId, 0, NULL);
ret = SQLBindParameter(stmt, 5, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, &id, 0, NULL);

ret = SQLExecute(stmt);

```

```

        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
std::cout<<"Невдалося змінити бронювання."<< std::endl;
        }
        else {
std::cout<<"Бронювання з ID "<< id <<" змінено:\n"
<<"ID гостя: "<<guestId<<"\n"
<<"Дата заїзду: "<<arrivalDateStr<<"\n"
<<"Дата виїзду: "<<checkoutDateStr<<"\n"
<<"ID кімнати: "<<roomId<<"\n"
<<std::endl;
        }

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

// Допоміжна функція для розбору рядка з датою
std::tm* parseDate(const std::string&dateStr)
{
std::tm* date = new std::tm();
std::stringstream ss(dateStr);
    ss >>std::get_time(date, "%Y-%m-%d");
    return date;
}

void removeBooking(int id, SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;

    // Звільнення кімнати
    SQLHSTMT updateRoomStmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &updateRoomStmt);
    ret = SQLPrepare(updateRoomStmt, (SQLCHAR*)"UPDATE
[dbo].[Rooms] SET [available] = 1 WHERE [id_room] IN (SELECT [room_id] FROM
[dbo].[Booking] WHERE [id] = ?)", SQL_NTS);
    ret = SQLBindParameter(updateRoomStmt, 1, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER, 0, 0, &id, 0, NULL);
    ret = SQLExecute(updateRoomStmt);

    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
std::cout<<"Помилка при звільненні кімнати."<< std::endl;
    }
    else {

```

```

std::cout<<"Кімната, пов'язана з бронюванням "<< id <<" , звільнена."<<
std::endl;

// Очищеннябуферапередзвільненнямресурсів
while (SQLMoreResults(updateRoomStmt) == SQL_SUCCESS) {}

// Видаленнябронювання
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
ret = SQLPrepare(stmt, (SQLCHAR*)"DELETE FROM
[dbo].[Booking] WHERE [id] = ?", SQL_NTS);

ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
SQL_C_SLONG, SQL_INTEGER, 0, 0, &id, 0, NULL);

ret = SQLExecute(stmt);

if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
std::cout<<"Невдалосявидалитибронювання."<<std::endl;
}
else {
std::cout<<"Бронюванняз ID "<<id<<"видалено."<<std::endl;
}

// Очищеннябуферапередзвільненнямресурсів
while (SQLMoreResults(stmt) == SQL_SUCCESS) {}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

SQLFreeHandle(SQL_HANDLE_STMT, updateRoomStmt);
}
void readAllBookings(SQLHDBC dbc)
{
SQLRETURN ret;
SQLHSTMT stmt;
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
ret = SQLExecDirect(stmt, (SQLCHAR*)"SELECT * FROM
[dbo].[Booking]", SQL_NTS);

if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
// Отриманнярезультатівзапиту
while (SQLFetch(stmt) == SQL_SUCCESS) {
int id, guestId, roomId;

```

```

std::string arrivalDate, checkoutDate;

// Отримання значень полів з результатів запиту
SQLGetData(stmt, 1, SQL_C_SLONG, &id, 0, NULL);
SQLGetData(stmt, 2, SQL_C_SLONG, &guestId, 0, NULL);
SQLGetData(stmt, 3, SQL_C_CHAR, &arrivalDate[0], sizeof(arrivalDate),
NULL);
SQLGetData(stmt, 4, SQL_C_CHAR, &checkoutDate[0],
sizeof(checkoutDate), NULL);
SQLGetData(stmt, 5, SQL_C_SLONG, &roomId, 0, NULL);

// Виведення отриманих значень
std::cout<<"ID бронювання: "<< id << std::endl;
std::cout<<"ID гостя: "<<guestId<< std::endl;
std::cout<<"Дата заїзду: "<<arrivalDate<< std::endl;
std::cout<<"Дата виїзду: "<<checkoutDate<< std::endl;
std::cout<<"ID кімнати: "<<roomId<< std::endl;
std::cout<< std::endl;
    }
}
else {
std::cout<<"Помилка при виконанні запиту."<<std::endl;
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void UpdatesBookings(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;

    // Зчитування всіх бронювань з таблиці
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    ret = SQLExecDirect(stmt, (SQLCHAR*)"SELECT [id], [OutDate]
FROM [dbo].[Booking]", SQL_NTS);

    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
std::cout<<"Помилка при зчитуванні бронювань."<< std::endl;
    }
    else {
        // Отримання результатів
        SQLLEN indicator;
        SQLCHAR outDate[20];
        SQLINTEGER id;

```

```

        ret = SQLBindCol(stmt, 1, SQL_C_SLONG, &id, sizeof(id),
&indicator);

        ret = SQLBindCol(stmt, 2, SQL_C_CHAR, outDate,
sizeof(outDate), &indicator);

        while (SQLFetch(stmt) == SQL_SUCCESS) {
            // Парсинг дати
            std::string outDateString(reinterpret_cast<char*>(outDate));
            std::tm outTime = {};
            std::istringstream ss(outDateString);
            ss >> std::get_time(&outTime, "%Y-%m-%d");

            if (ss.fail()) {
                std::cout<<"Помилка при парсингу дати в ізд для бронювання з ID "<< id <<
std::endl;
                continue;
            }

            // Отримання поточної дати
            auto now = std::chrono::system_clock::now();
            std::time_t nowTime = std::chrono::system_clock::to_time_t(now);
            std::tm nowTM = {};
            localtime_s(&nowTM, &nowTime);

            // Порівняння дат
            if (std::mktime(&nowTM) > std::mktime(&outTime)) {
                // Звільнення кімнати
                SQLHSTMT updateRoomStmt;
                ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc,
&updateRoomStmt);

                ret = SQLPrepare(updateRoomStmt, (SQLCHAR*)"UPDATE
[dbo].[Rooms] SET [available] = 1 WHERE [id_room] IN (SELECT [room_id] FROM
[dbo].[Booking] WHERE [id] = ?)", SQL_NTS);
                ret = SQLBindParameter(updateRoomStmt, 1,
SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &id, 0, NULL);
                ret = SQLExecute(updateRoomStmt);

                if (ret != SQL_SUCCESS && ret !=
SQL_SUCCESS_WITH_INFO) {
                    std::cout<<"Помилка при звільненні кімнати для бронювання з ID "<< id <<
std::endl;
                }
            }
            else {

```

```

std::cout<<"Кімната, пов'язана з бронюванням "<< id <<" , звільнена."<<
std::endl;

// Очищеннябуферапередзвільненнямресурсів
while (SQLMoreResults(updateRoomStmt) ==

SQL_SUCCESS) {}

// Видаленнябронювання
ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc,
&stmt);

ret = SQLPrepare(stmt, (SQLCHAR*)"DELETE FROM
[dbo].[Booking] WHERE [id] = ?", SQL_NTS);
ret = SQLBindParameter(stmt, 1,
SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &id, 0, NULL);
ret = SQLExecute(stmt);

if (ret != SQL_SUCCESS && ret !=
SQL_SUCCESS_WITH_INFO) {
std::cout<<"Невдалосявидалитибронювання з ID "<< id << std::endl;
}
else {
std::cout<<"Бронювання з ID "<< id <<" видалено."<< std::endl;
}

// Очищеннябуферапередзвільненнямресурсів
while (SQLMoreResults(stmt) == SQL_SUCCESS) {}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

SQLFreeHandle(SQL_HANDLE_STMT, updateRoomStmt);
}
else {
std::cout<<"Датавиїздудлябронювання з ID "<< id <<" щененастала."<<
std::endl;
}
}
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}
};

```

A.3 Текст файлу Header Guests.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <sql.h>
#include <sqlext.h>

using namespace std;

class Guest {
public:
    int id;
    string name;
    Guest() {
        id = 0;
        name = "";
    }

    Guest(int i, string n) {
        id = i;
        name = n;
    }

    void addGuest(SQLHDBC dbc) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        string guestName;
        cout<<"Введіть ім'я гостя: ";
        getline(cin, guestName);

        string query = "INSERT INTO [dbo].[Guest] (Guest_name) VALUES
(?)";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовці запиту для додавання гостя.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        }
    }
};
```

```

return;
    }

    SQLLEN param1 = SQL_NTS;

    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)guestName.c_str(), 0, &param1);

    ret = SQLExecute(stmt);
    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
cout<<"Гостя успішно додано!"<<endl;
    }
else {
cout<<"Помилка при виконанні запиту для додавання гостя.\n";
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void deleteGuest(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    int guestId;
cout<<"Введіть ідентифікатор гостя: ";
cin>>guestId;

    string query = "DELETE FROM [dbo].[Guest] WHERE id = ?";

    ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
cout<<"Помилка при підготовці запиту для видалення гостя.\n";
SQLFreeHandle(SQL_HANDLE_STMT, stmt);
return;
    }

    SQLLEN param1 = SQL_NTS;

    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&guestId, 0, &param1);

    ret = SQLExecute(stmt);
    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {

```

```

cout<<"Гостя успішно видалено!"<<endl;
    }

else {
cout<<"Помилка при виконанні запиту для видалення гостя.\n";
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void editGuest(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    int guestId;
    cout<<"Введіть ідентифікатор гостя: ";
    cin>>guestId;

    string guestName;
    cout<<"Введіть нове ім'я гостя: ";
    cin.ignore(); // Пропускаємо перенос рядка після вводу guestId
    getline(cin, guestName);

    string query = "UPDATE [dbo].[Guest] SET Guest_name = ? WHERE
id = ?";

    ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        cout<<"Помилка при підготовці запиту для редагування гостя.\n";
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return;
    }

    SQLLEN param1 = SQL_NTS;
    SQLLEN param2 = SQL_NTS;

    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)guestName.c_str(), 0, &param1);
    ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&guestId, 0, &param2);

    ret = SQLExecute(stmt);

```

```

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Інформацію про гостя успішнооновлено!"<<endl;
        }
    else {
        cout<<"Помилка при виконаннізапиту для редагування гостя.\n";
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void showAllGuests(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    string query = "SELECT * FROM [dbo].[Guest]";

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        cout<<"Помилка при виконаннізапиту для отриманняінформації про всіх
        ростей.\n";
    }
    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    return;
}

    int id;
    SQLCHAR guestName[256];
    SQLLEN ind1, ind2;

    cout<<"ID\tGuest Name"<<endl;
    cout<<"-----"<<endl;

    while (SQLFetch(stmt) == SQL_SUCCESS) {
        SQLGetData(stmt, 1, SQL_C_LONG, &id, sizeof(id), &ind1);
        SQLGetData(stmt, 2, SQL_C_CHAR, guestName, sizeof(guestName), &ind2);

        if (ind1 == SQL_NULL_DATA || ind2 == SQL_NULL_DATA) {
            cout<<"NULL\tNULL"<<endl;
        }
        else {
            cout<< id <<"\t"<<guestName<<endl;
        }
    }
}

```

```

    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

};

```

A.4 Текст файла Header Invent.h

```

#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <sql.h>
#include <sqlext.h>

using namespace std;

class Product {
public:
    int id;
    string name;
    double price;

    Product(int i, string n, double p) {
        id = i;
        name = n;
        price = p;
    }
};

class Inventory {
public:

    void addProduct(SQLHDBC dbc, string name, double price) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    }
};

```

```

        string query = "INSERT INTO [dbo].[Product] (Name_posluga,
Price) VALUES (?, ?)";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовцізапиту для додавання продукту.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
            return;
        }

        SQLLEN param1 = SQL_NTS;
        SQLLEN param2 = SQL_NTS;

        ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)name.c_str(), 0, &param1);
        ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_FLOAT, 0, 0, (SQLPOINTER)&price, 0, &param2);

        ret = SQLExecute(stmt);
        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Продукт успішно додано!"<<endl;
        }
        else {
            cout<<"Помилка при виконаннізапиту для додавання продукту.\n";
        }

        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

    void deleteProduct(SQLHDBC dbc, int id) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        string query = "DELETE FROM [dbo].[Product] WHERE id = ?";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовцізапиту для видалення продукту.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
            return;
        }
    }

```

```

        SQLLEN param1 = SQL_NTS;

        ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&id, 0, &param1);

        ret = SQLExecute(stmt);
        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Продукт успішно видалено!"<<endl;
        }
        else {
            cout<<"Помилка при виконанні запиту для видалення продукту.\n";
        }

        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

    void editProduct(SQLHDBC dbc, int id, string name, double price) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        string query = "UPDATE [dbo].[Product] SET Name_posluga = ?,
Price = ? WHERE id = ?";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовці запиту для редагування продукту.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
            return;
        }

        SQLLEN param1 = SQL_NTS;
        SQLLEN param2 = SQL_NTS;
        SQLLEN param3 = SQL_NTS;

        ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)name.c_str(), 0, &param1);
        ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_FLOAT, 0, 0, (SQLPOINTER)&price, 0, &param2);
        ret = SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&id, 0, &param3);

        ret = SQLExecute(stmt);
    }

```

```

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Продукту успішно оновлено!"<<endl;
        }
    }
    else {
        cout<<"Помилка при виконанні запиту для редагування продукту.\n";
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void showAllProducts(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    string query = "SELECT * FROM [dbo].[Product]";

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        cout<<"Помилка при виконанні запиту для отримання всіх продуктів.\n";
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }
    return;
}

    int id;
    SQLCHAR name[256];
    double price;
    SQLLEN ind1, ind2, ind3;

    cout<<"ID\tName\tPrice"<<endl;
    cout<<"-----"<<endl;

    while (SQLFetch(stmt) == SQL_SUCCESS) {
        SQLGetData(stmt, 1, SQL_C_LONG, &id, sizeof(id), &ind1);
        SQLGetData(stmt, 2, SQL_C_CHAR, name, sizeof(name), &ind2);
        SQLGetData(stmt, 3, SQL_C_DOUBLE, &price, sizeof(price), &ind3);

        if (ind1 == SQL_NULL_DATA || ind2 == SQL_NULL_DATA || ind3
== SQL_NULL_DATA) {
            cout<<"NULL\tNULL\tNULL"<<endl;
        }
        else {
            cout<< id <<"\t"<< name <<"\t"<< price <<endl;
        }
    }
}

```

```

        }
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

};

```

A.5 Текст файла Header Manag.h

```

#pragma once
#include <iostream>
#include <vector>
#include <string>
#include <sql.h>
#include <sqlext.h>
using namespace std;

class Employee {
public:
    int id;
    string name;
    string position;
    string schedule;
    Employee(int i, string n, string p, string s) {
        id = i;
        name = n;
        position = p;
        schedule = s;
    }
};

class PersonnelManager {
private:
    vector<Employee>employees;

public:

    void addEmployee(SQLHDBC dbc, const string& name, const string&
position, const string& schedule) {

```

```

        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        // Формування запиту для додавання нового працівника
        string query = "INSERT INTO [dbo].[Employeees] (name, position,
schedule) VALUES (?, ?, ?)";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовці запиту.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
            return;
        }

        SQLLEN param1 = SQL_NTS;
        SQLLEN param2 = SQL_NTS;
        SQLLEN param3 = SQL_NTS;

        ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)name.c_str(), 0, &param1);
        ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)position.c_str(), 0, &param2);
        ret = SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)schedule.c_str(), 0, &param3);

        ret = SQLExecute(stmt);
        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Інформація про нового працівника успішно додана!"<<endl;
        }
        else {
            cout<<"Помилка при виконанні запиту.\n";
        }

        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

    void removeEmployee(SQLHDBC dbc, int index) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        // Формування запиту для видалення співробітника за індексом

```

```

        string query = "DELETE FROM [dbo].[Employeees] WHERE id_employee = " +
to_string(index + 1); // Зверніть увагу, що тут індекс збігається з
значенням id_employee

        ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
            cout<<"Співробітнику успішно видалений!"<<endl;
        }
        else {
            cout<<"Помилка при виконанні запиту.\n";
        }

        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

    void editEmployee(SQLHDBC dbc, int id, const string& name, const
string& position, const string& schedule) {
        SQLRETURN ret;
        SQLHSTMT stmt;
        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

        // Формування запиту для оновлення співробітника за id
        string query = "UPDATE [dbo].[Employeees] SET name = ?, position = ?,
schedule = ? WHERE id_employee= ?";

        ret = SQLPrepare(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);
        if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
            cout<<"Помилка при підготовці запиту.\n";
            SQLFreeHandle(SQL_HANDLE_STMT, stmt);
            return;
        }

        SQLLEN param1 = SQL_NTS;
        SQLLEN param2 = SQL_NTS;
        SQLLEN param3 = SQL_NTS;
        SQLLEN param4 = id;

        ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)name.c_str(), 0, &param1);
        ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)position.c_str(), 0, &param2);

```

```

        ret = SQLBindParameter(stmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)schedule.c_str(), 0, &param3);
        ret = SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, &param4, sizeof(SQLLEN), &param4);

        ret = SQLExecute(stmt);
        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
cout<<"Співробітникуспішновідредагований!"<<endl;
        }
        else {
cout<<"Помилкапривиконаннізапиту.\n";
        }

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

void displayEmployees(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формуваннязапитудляотриманнявсіхспівробітників
    string query = "SELECT * FROM [dbo].[Employyes]";

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
cout<<"Співробітники:"<<endl;

        // Перебіррезультатівзапиту
while (SQLFetch(stmt) == SQL_SUCCESS) {
            SQLINTEGER id;
            SQLCHAR name[256];
            SQLCHAR position[256];
            SQLCHAR schedule[256];

            // Отриманнязначеньспівробітника
SQLGetData(stmt, 1, SQL_INTEGER, &id, sizeof(id), NULL);
SQLGetData(stmt, 2, SQL_CHAR, name, sizeof(name), NULL);
SQLGetData(stmt, 3, SQL_CHAR, position, sizeof(position), NULL);
SQLGetData(stmt, 4, SQL_CHAR, schedule, sizeof(schedule), NULL);

```

```

        employees.push_back(Employee(id,      (char*)name,      (char*)position,
(char*)schedule));

    cout<<" "<< id <<" . "<< name <<" ("<< position <<)" "<<endl;
    cout<<" Графікроботи: "<< schedule <<endl;
}

    }

else {
    cout<<"Помилка при виконаннізапиту.\n";
}

SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

};

```

A.6 Текст файлу Header Room.h

```

#pragma once
#include <iostream>
#include <sql.h>
#include <sqlext.h>
#include <string>
#include <sstream>

class Room {
public:
    Room(int id, std::string type, int capacity, bool available) {
        this->id = id;
        this->type = type;
        this->capacity = capacity;
        this->available = available;
    }

    int getId() const {
        return id;
    }

    std::string getType() const {
        return type;
    }
}

```

```

        int getCapacity() const {
            return capacity;
        }

        bool isAvailable() const {
            return available;
        }

        ~Room() {}

private:
    int id;
    std::string type;
    int capacity;
    bool available;
};

class roomSystem {
public:
    void addRoom(SQLHDBC dbc) {
        SQLRETURNret;

        // Зчитування даних для вставки
        std::string roomType;
        int roomCapacity;
        bool roomAvailable;

        std::cout<<"Введіть тип кімнати: ";
        std::cin>>roomType;

        std::cout<<"Введіть місткість кімнати: ";
        std::cin>>roomCapacity;

        std::cout<<"Введіть наявність кімнати (0 - ні, 1 - так): ";
        std::cin>>roomAvailable;

        // Вставка даних у таблицю
        SQLHSTMT stmt;
        SQLCHAR query[256];

        snprintf((char*)query, sizeof(query), "INSERT INTO Rooms (type,
        capacity, available) VALUES ('%s', %d, %d)", roomType.c_str(), roomCapacity,
        roomAvailable ? 1 : 0);
    }
};

```

```

        ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
        ret = SQLExecDirect(stmt, query, SQL_NTS);

        if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
std::cout<<"Кімнату успішно додано."<< std::endl;
        }
        else {
std::cout<<"Помилка при додаванні."<< std::endl;
        }

        // Звільнення ресурсів
SQLFreeHandle(SQL_HANDLE_STMT, stmt);
    }

void displayAllRooms(SQLHDBC dbc) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    ret = SQLExecDirect(stmt, (SQLCHAR*)"SELECT * FROM
[dbo].[Rooms]", SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        // Прив'язка стовпців до змінних
        int id, capacity;
        char type[51];
        SQLLEN typeLen;
        bool available;

        SQLBindCol(stmt, 1, SQL_C_SLONG, &id, 0, NULL);
        SQLBindCol(stmt, 2, SQL_C_CHAR, type, sizeof(type), &typeLen);
        SQLBindCol(stmt, 3, SQL_C_SLONG, &capacity, 0, NULL);
        SQLBindCol(stmt, 4, SQL_C_BIT, &available, 0, NULL);

        // Отримання результатів запиту
        while (SQLFetch(stmt) == SQL_SUCCESS) {
            // Виведення отриманих значень
std::cout<<"Номер кімнати: "<< id << std::endl;
std::cout<<"Тип: "<< std::string(type, typeLen) << std::endl;
std::cout<<"Вмістимість: "<< capacity << std::endl;
std::cout<<"Доступність: "<< (available ? "Доступна" : "Недоступна")
<< std::endl;
std::cout<< std::endl;

```

```

        }
    }
    else {
        std::cout<<"Помилка при виконанні запиту."<<std::endl;
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void findAvailableRoom(SQLHDBC dbc, const std::string&roomType,
int roomCapacity) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту з умовою на тип кімнати та вмістимість
    std::string query = "SELECT * FROM [dbo].[Rooms] WHERE type = '" +
roomType + "' AND capacity >= " + std::to_string(roomCapacity) + " AND
available = 1";

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        // Прив'язка стовпців до змінних
        int id, capacity;
        char type[51];
        SQLLEN typeLen;
        bool available;

        SQLBindCol(stmt, 1, SQL_C_SLONG, &id, 0, NULL);
        SQLBindCol(stmt, 2, SQL_C_CHAR, type, sizeof(type), &typeLen);
        SQLBindCol(stmt, 3, SQL_C_SLONG, &capacity, 0, NULL);
        SQLBindCol(stmt, 4, SQL_C_BIT, &available, 0, NULL);

        // Отримання результатів запиту
        if (SQLFetch(stmt) == SQL_SUCCESS) {
            // Виведення отриманих значень
            std::cout<<"Знайдена вільна кімната:"<< std::endl;
            std::cout<<"Номер кімнати: "<< id << std::endl;
            std::cout<<"Тип: "<< std::string(type, typeLen) << std::endl;
            std::cout<<"Вмістимість: "<< capacity << std::endl;
            std::cout<< std::endl;
        }
    }
    else {

```

```

        std::cout<<"Немає вільних кімнат,          які відповідають заданим критеріям."<<
std::endl;

        }

    }

    else {

std::cout<<"Помилка при виконанні запиту."<< std::endl;

    }

SQLFreeHandle(SQL_HANDLE_STMT, stmt);

}

void getRoomById(SQLHDBC dbc, int roomId) {

    SQLRETURN ret;

    SQLHSTMT stmt;

    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту з умовою на ID кімнати

    std::string query = "SELECT * FROM [dbo].[Rooms] WHERE id_room = " +
std::to_string(roomId);

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {

        // Прив'язка стовпців до змінних

        int id, capacity;

        char type[51];

        SQLENN typeLen;

        bool available;

        SQLBindCol(stmt, 1, SQL_C_SLONG, &id, 0, NULL);

        SQLBindCol(stmt, 2, SQL_C_CHAR, type, sizeof(type), &typeLen);

        SQLBindCol(stmt, 3, SQL_C_SLONG, &capacity, 0, NULL);

        SQLBindCol(stmt, 4, SQL_C_BIT, &available, 0, NULL);

        // Отримання результатів запиту

        if (SQLFetch(stmt) == SQL_SUCCESS) {

            // Виведення отриманих значень

            std::cout<<"Інформація про кімнату за номером: "<<roomId<<":"<<std::endl;

            std::cout<<"Тип: "<< std::string(type, typeLen) << std::endl;

            std::cout<<"Вмістимість: "<< capacity << std::endl;

            std::cout<<"Доступність: "<< (available ? "Доступна" : "Недоступна")

<< std::endl;

            std::cout<< std::endl;

        }

    }

```

```

        else {
std::cout<<"Кімната з ID "<<roomId<<" не знайдена."<< std::endl;
        }
    }
    else {
std::cout<<"Помилка при виконанні запиту."<< std::endl;
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void closeRoomAvailability(SQLHDBC dbc, int roomId) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту для оновлення доступності кімнати
    std::string query = "UPDATE [dbo].[Rooms] SET available = 0 WHERE
id_room = " + std::to_string(roomId);

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
std::cout<<"Кімнату з номером          "<<roomId<<"          успішно закрито."<<
std::endl;
    }
    else {
std::cout<<"Помилка при виконанні запиту."<< std::endl;
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}

void openRoomAvailability(SQLHDBC dbc, int roomId) {
    SQLRETURN ret;
    SQLHSTMT stmt;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);

    // Формування запиту для оновлення доступності кімнати
    std::string query = "UPDATE [dbo].[Rooms] SET available = 1 WHERE
id_room = " + std::to_string(roomId);

    ret = SQLExecDirect(stmt, (SQLCHAR*)query.c_str(), SQL_NTS);

    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {

```

```

        std::cout<<"Кімнату заномером " << roomId <<" успішно відкрито." <<
std::endl;
    }
    else {
        std::cout<<"Помилка при виконанні запиту." << std::endl;
    }

    SQLFreeHandle(SQL_HANDLE_STMT, stmt);
}
};

```

A.7 Текст файлу Header Project_hotel.h

```

// CMakeProject1.h
: включає файл для стандартних системних включаємх файлів
// или включаемые файлы для конкретного проекта.

#pragma once
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <iterator>
#include <Windows.h>
#include <cstdint>
#include <sql.h>
#include <sqlext.h>
#include "Booking.h"
#include <cstdlib>
#include "Room.h"
#include "BM.h"
#include "Manag.h"
#include "Invent.h"
#include "Guests.h"
#include <chrono>

// TODO: установите здесь ссылки на дополнительные заголовки,
требуемые для программы.

```

A.8 Текст файлу Main.cpp

```
// CMakeProject1.cpp: Визначає точку входу для додатка.
//
#include "Project_hotel.h"

using namespace std;

void Guests_menu(SQLHANDLE dbc)
{
    SQLHANDLE gdbc;
    gdbc = dbc;
    Guest guest;
    int SateGuestMenu = 0;
    while (SateGuestMenu != 5) {
        cout<<"Меню:\n"
        <<"1. Додати інформацію про нового гостя.\n"
        <<"2. Редагувати інформацію про гостя.\n"
        <<"3. Видалити інформацію про гостя.\n"
        <<"4. Показати всі дані про гостей.\n"
        <<"5. Повернутись назад.\n";

        cin>>SateGuestMenu;

        switch (SateGuestMenu) {
            case 1:
                system("cls");
                guest.addGuest(gdbc);
                system("pause");
                system("cls");
                break;
            case 2:
                system("cls");
                guest.editGuest(gdbc);
                system("pause");
                system("cls");
                break;
            case 3:
                system("cls");
                guest.deleteGuest(gdbc);
                system("pause");
                system("cls");
```

```

break;
        case 4:
            system("cls");
            guest.showAllGuests(gdbc);
            system("pause");
            system("cls");
        break;
        case 5:
            system("cls");
        return;
        default:
            cout<<"Невірний вибір. Спробуйте ще раз.\n";
            break;
    }
}

void Booking_menu(SQLHANDLE dbc)
{
    SQLHANDLE bdbc;
    bdbc = dbc;
    BookingSystembooking;
    string arrivalDate;
    string checkoutDate;
    int roomId, StateBookMenu = 0, bookingId, Gest_id;

    while (StateBookMenu != 5) {
        cout<<"Меню:\n"
        <<"1. Створити нове бронювання.\n"
        <<"2. Редагувати бронювання.\n"
        <<"3. Видалити бронювання.\n"
        <<"4. Показати всі дії бронювання.\n"
        <<"5. Повернутись назад.\n";

        cin>>StateBookMenu;

        switch (StateBookMenu) {
            case 1:
                system("cls");
                cout<<"1. Створити нове бронювання.\n";
                cout<<"Введіть дату заїзду (рік-місяць-день): ";
                cin>>arrivalDate;
                cout<<"Введіть дату виїзду (рік-місяць-день): ";
                cin>>checkoutDate;
            }
        }
    }
}

```

```

cout<<"Введіть ID кімнати: ";
cin>>roomId;
cout<<"Введіть ID гостя: ";
cin>>Gest_id;

    system("cls");
booking.addBooking(arrivalDate, checkoutDate, roomId,Gest_id, bdbc);
    system("pause");
    system("cls");

break;

    case 2:
        system("cls");
cout<<"2. Редагувати бронювання.\n";
cout<<"Введіть Id бронювання яке бажаєте змінити: ";
cin>>bookingId;
booking.editBooking(bookingId, bdbc);
    system("pause");
    system("cls");

break;

    case 3:
        system("cls");
cout<<"3. Видалити бронювання.\n";
cout<<"Введіть Id бронювання яке бажаєте видалити : ";
cin>>bookingId;
booking.removeBooking(bookingId, bdbc);
    system("pause");
    system("cls");

break;

    case 4:
        system("cls");
cout<<"4. Показати всі дійсні бронювання.\n";
booking.readAllBookings(bdbc);
    system("pause");
    system("cls");

break;

    case 5:
        system("cls");

return;

    default:
cout<<"Невірний вибір. Спробуйте ще раз.\n";
break;
}

}

};

```

```

void Room_menu(SQLHANDLE dbc)
{
    SQLHANDLE rdbc;
    rdbc = dbc;
    roomSystemroom;
    int StateRoomMenu = 0, roomCapacity, roomId;
    string roomType;

    while (StateRoomMenu != 7) {
        cout<<"Меню:\n"
        <<"1. Вивести інформацію про всі кімнати.\n"
        <<"2. Знайти вільну кімнату.\n"
        <<"3. Інформація про кімнату.\n"
        <<"4. Закрити кімнату.\n"
        <<"5. Відкрити кімнату.\n"
        <<"6. Додати нову кімнату.\n"
        <<"7. Повернутись назад.\n";

        cin>>StateRoomMenu;

        switch (StateRoomMenu) {
            case 1:
                system("cls");
                cout<<"1. Вивести інформацію про всі кімнати.\n";
                room.displayAllRooms(rdbc);

                system("pause");
                system("cls");
                break;
            case 2:
                system("cls");
                cout<<"2. Знайти вільну кімнату.\n";
                cout<<"Введіть тип кімнати: ";
                cin>>roomType;
                cout<<"Введіть вміст кімнати: ";
                cin>>roomCapacity;
                room.findAvailableRoom(rdbc, roomType, roomCapacity);
                system("pause");
                system("cls");
                break;
            case 3:
                system("cls");
                cout<<"3. Інформація про кімнату.\n";

```

```

cout<<"Введіть номер кімнати: ";
cin>>roomId;
room.getRoomById(rdbc, roomId);
    system("pause");
    system("cls");
break;
    case 4:
        system("cls");
cout<<"4. Закрити кімнату.\n";
cout<<"Введіть номер кімнати: ";
cin>>roomId;
room.closeRoomAvailability(rdbc, roomId);
    system("pause");
    system("cls");
break;
    case 5:
        system("cls");
cout<<"5. Відкрити кімнату.\n";
cout<<"Введіть номер кімнати: ";
cin>>roomId;
room.openRoomAvailability(rdbc, roomId);
    system("pause");
    system("cls");
break;
    case 6:
        system("cls");
"6. Додати нову кімнату.\n";
cout<<"1. Вивести інформацію про всі кімнати.\n";
room.addRoom(rdbc);
    system("pause");
    system("cls");
break;
    case 7:
        system("cls");
return;

    default:
        system("cls");
cout<<"Невірний вибір. Спробуйте ще раз.\n";
break;
}
}
};

```

```

void Bill_menu(SQLHANDLE dbc)
{
    SQLHANDLE bdbc;
    bdbc = dbc;
    BillManagement billManagement;
    double amount, newAmount;
    int StateBillMenu = 0, guestId, billId;

    while (StateBillMenu != 6) {
        cout<<"Меню:\n"
        <<"1. Створити новий рахунок.\n"
        <<"2. Видалити рахунок.\n"
        <<"3. Редагувати рахунок.\n"
        <<"4. Показати всі рахунки.\n"
        <<"5. Надіслати рахунок гостю.\n"
        <<"6. Повернутись назад.\n";

        cin>>StateBillMenu;

        switch (StateBillMenu) {
            case 1:
                system("cls");
                cout<<"1. Створити новий рахунок.\n";
                cout<<"Введіть Id гостя: ";
                cin>>guestId;
                cout<<"Введіть суму яку потрібно додати до рахунку: ";
                cin>>amount;
                billManagement.addBill(bdbc, guestId, amount);
                system("pause");
                system("cls");
                break;

            case 2:
                system("cls");
                cout<<"2. Видалити рахунок.\n";
                cout<<"Введіть id рахунку: ";
                cin>>billId;
                billManagement.deleteBill(bdbc, billId);
                system("pause");
                system("cls");
                break;

            case 3:
                system("cls");
                cout<<"3. Редагувати рахунок.\n";

```

```

cout<<"Введіть id рахунку: ";
cin>>billId;
cout<<"Введітьзміненийрахунок";
cin>>newAmount;
billManagement.editBill(bdbc, billId, newAmount);
    system("pause");
    system("cls");
break;
    case 4:
        system("cls");
cout<<"4. Показативсірахунки.\n";
billManagement.showAllBills(bdbc);
    system("pause");
    system("cls");
break;
    case 5:
        system("cls");
cout<<"5. Надіслатирахунокгостю.\n";
cout<<"Введітьidрахунку: ";
cin>>billId;
billManagement.sendBillToGuest(bdbc, billId);
    system("pause");
    system("cls");
break;
    case 6:
        system("cls");
return;
    default:
        system("cls");
cout<<"Невірнийвибір. Спробуйтещераз.\n";
break;
}
}
};
void Employee_menu(SQLHANDLE dbc)
{
    SQLHANDLE edbc;
edbc = dbc;
PersonnelManagerpersonnelManager;
    string newName;
    string newPosition;
    string newSchedule;
    string name, position, schedule;

```

```

int StateEmployeeMenu = 0, employeeIndex;

while (StateEmployeeMenu != 5) {
cout<<"Меню:\n"
<<"1. Додати інформацію про нового працівника.\n"
<<"2. Видалити інформацію про працівника.\n"
<<"3. Редагувати інформацію про працівника.\n"
<<"4. Показати всю інформацію про працівників.\n"
<<"5. Повернутись назад.\n";

cin>>StateEmployeeMenu;

switch (StateEmployeeMenu) {
case 1:
    system("cls");
cout<<"1. Додати інформацію про нового працівника.\n";
cout<<"Введіть ім'я нового працівника: ";
getline(cin, name);

cout<<"Введіть посаду нового працівника: ";
getline(cin, position);

cout<<"Введіть графік роботи нового працівника: ";
getline(cin, schedule);

personnelManager.addEmployee(edbc, name, position, schedule);
    system("pause");
    system("cls");
break;
case 2:
    system("cls");
cout<<"2. Видалити інформацію про працівника.\n";
cout<<"Введіть Id співробітника: ";
cin>>employeeIndex;
personnelManager.removeEmployee(edbc, employeeIndex);
    system("pause");
    system("cls");
break;
case 3:
    system("cls");
cout<<"3. Редагувати інформацію про працівника.\n";
cout<<"Введіть індекс співробітника, якого потрібно відредагувати: ";
cin>>employeeIndex;

```

```

cout<<"Введіть нове ім'я співробітника: ";
getline(cin, newName);

cout<<"Введіть нову посаду співробітника: ";
getline(cin, newPosition);

cout<<"Введіть новий графік роботи співробітника: ";
getline(cin, newSchedule);

    personnelManager.editEmployee(edbc,          employeeIndex,          newName,
newPosition, newSchedule);
        system("pause");
        system("cls");
    break;

    case 4:
        system("cls");
    cout<<"4. Показати всю інформацію про працівників.\n";
    personnelManager.displayEmployees(edbc);
        system("pause");
        system("cls");
    break;

    case 5:
        system("cls");
    return;

    default:
        system("cls");
    cout<<"Невірний вибір. Спробуйте ще раз.\n";
    break;
}

};

void Invent_menu(SQLHANDLE dbc)
{
    SQLHANDLE idbc;
    Inventory inventory;
    string name;
    double price;
    idbc = dbc;
    int StateInvMenu = 0, id;

    while (StateInvMenu != 5) {
    cout<<"Меню:\n"

```

```

<<"1. Додати нову послугу.\n"
<<"2. Видалити послугу.\n"
<<"3. Редагувати послуги.\n"
<<"4. Показати всі послуги.\n"
<<"5. Повернутись назад.\n";

cin>>StateInvMenu;

switch (StateInvMenu) {
    case 1:
        system("cls");
        cout<<"1. Додати нову послугу.\n";
        cout<<"Введіть назву продукту: ";
        getline(cin, name);
        cout<<"Введіть ціну продукту: ";
        cin>>price;
        inventory.addProduct(idbc, name, price);
        system("pause");
        system("cls");
        break;
    case 2:
        system("cls");
        cout<<"2. Видалити послугу.\n";
        cout<<"Введіть ID продукту, який потрібно видалити: ";
        cin>>id;
        inventory.deleteProduct(idbc, id);
        system("pause");
        system("cls");
        break;
    case 3:
        system("cls");
        cout<<"3. Редагувати послуги.\n";
        cout<<"Введіть ID продукту, який потрібно редагувати: ";
        cin>>id;
        cout<<"Введіть нове ім'я продукту: ";
        getline(cin, name);
        cout<<"Введіть нову ціну продукту: ";
        cin>>price;
        inventory.editProduct(idbc, id, name, price);
        system("pause");
        system("cls");
        break;
    case 4:

```

```

        system("cls");
    cout<<"4. Показати всі послуги.\n";
    inventory.showAllProducts(idbc);
        system("pause");
        system("cls");
    break;

    case 5:
        system("cls");
    return;

    default:
        system("cls");
    cout<<"Невірний вибір. Спробуйте ще раз.\n";
    break;
}

};

void menu(SQLHANDLE hdbc)
{
    SQLHANDLE dbc;
    dbc = hdbc;
    int StateMenu = 0;

    while (StateMenu != 7) {
    cout<<"Меню:\n"
    <<"1. Робота з бронюваннями.\n"
    <<"2. Робота з кімнатами.\n"
    <<"3. Робота з рахунками.\n"
    <<"4. Робота з працівниками.\n"
    <<"5. Робота з послугами готелю.\n"
    <<"6. Робота з відвідувачами.\n"
    <<"7. Вихід\n";

    cin>>StateMenu;

        switch (StateMenu) {
            case 1:
                system("cls");
                Booking_menu(dbc);
                StateMenu = 0;
                break;

            case 2:
                system("cls");
                Room_menu(dbc);

```

```

StateMenu = 0;
break;

        case 3:
            system("cls");
Bill_menu(dbc);
StateMenu = 0;
break;

        case 4:
            system("cls");
Employee_menu(dbc);
StateMenu = 0;
break;

        case 5:
            system("cls");
Invent_menu(dbc);
StateMenu = 0;
break;

        case 6:
            system("cls");
Guests_menu(dbc);
StateMenu = 0;
break;

        case 7:
            system("cls");
cout<<"Завершення роботи!\n";
return;

        default:
cout<<"Невірний вибір. Спробуйте ще раз.\n";
break;
    }
}

};

int main()
{
BookingSystemupdate;
SetConsoleOutputCP(1251);
SetConsoleCP(1251);
    SQLHANDLE henv;
    SQLHANDLE hdbc;
    SQLHANDLE hstmt;
    SQLRETURN retcode;
    // Ініціалізація середовища

```

```

retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
(SQLPOINTER)SQL_OV_ODBC3, SQL_IS_INTEGER);

// Створення підключення
retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
retcode = SQLDriverConnect(hdbc, NULL, (SQLCHAR*)"Driver={ODBC Driver
18 for SQL Server};Server=tcp:745435.database.windows.net,1433;Database=For
Hotel;Uid=Sgrims;Pwd=Lendra745435;Encrypt=yes;TrustServerCertificate=no;Conne
ction Timeout=30;", SQL_NTS, NULL, 0, NULL, SQL_DRIVER_COMPLETE);

// Перевірка статусу підключення
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
std::cout<<"Підключення успішне."<< std::endl;

// Виконання запиту
retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
retcode = SQLExecDirect(hstmt, (SQLCHAR*)"SELECT * FROM YourTable",
SQL_NTS);
update.UpdatesBookings(hdbc);
system("pause");
system("cls");
menu(hdbc);
// Звільнення ресурсів
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
SQLDisconnect(hdbc);
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, henv);
exit(0);
}
else {
std::cout<<"Помилка підключення."<< std::endl;
}

return 0;
};

```

ДОДАТОК Б
Слайди презентації

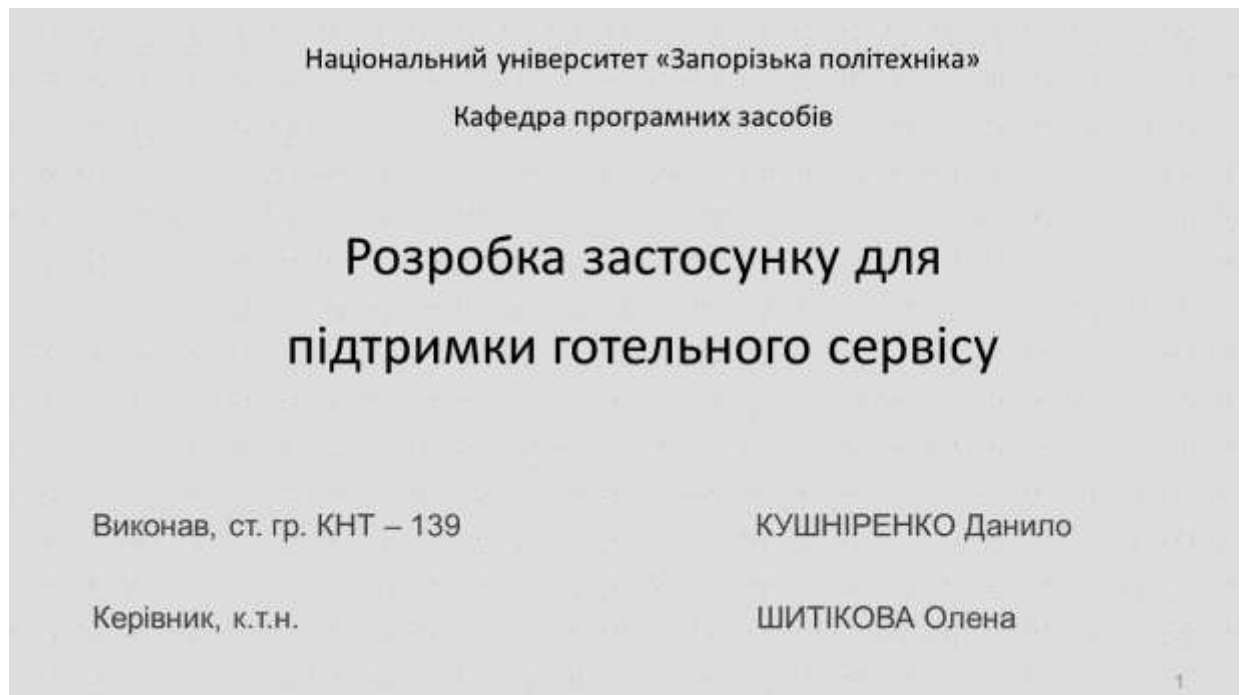


Рисунок Б.1 - Слайд 1

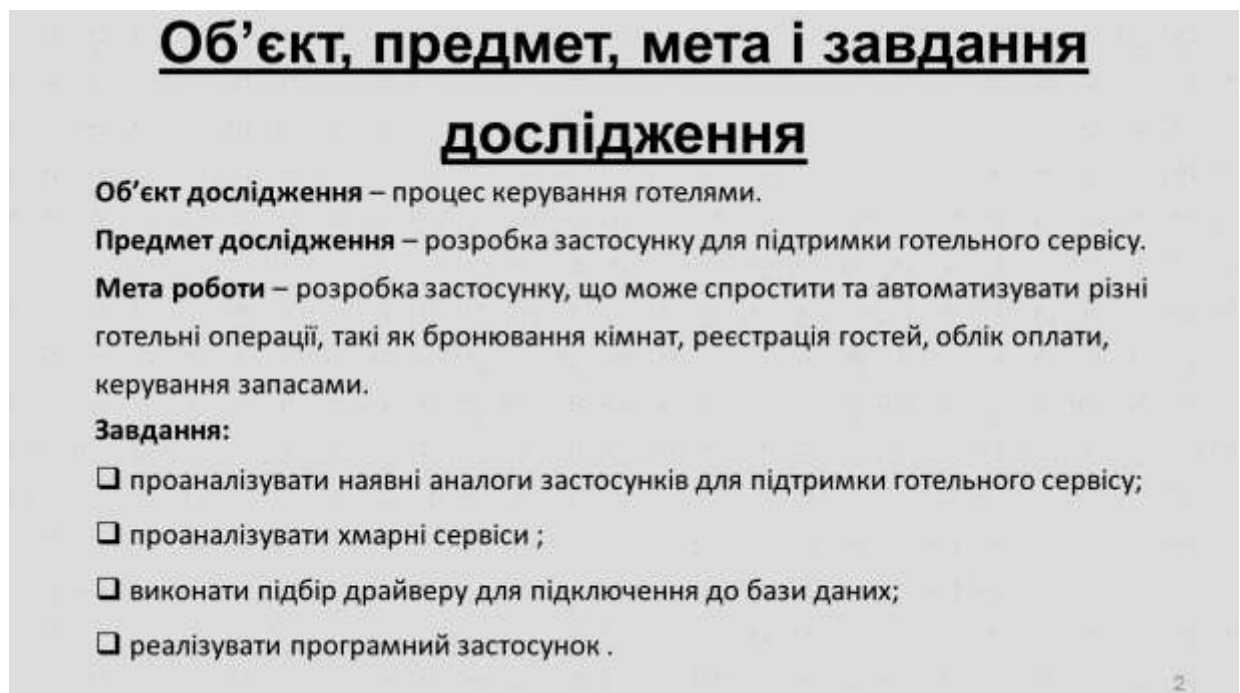


Рисунок Б.2 - Слайд 2

Актуальність теми дослідження

- ☐ автоматизація та оптимізація процесу підтримки готельного сервісу;
- ☐ збільшення продуктивності керування ресурсами;
- ☐ підвищення конкурентоспроможності.

3

Рисунок Б.3 - Слайд 3

АСУ для підтримки готельного сервісу

Порівняння платформ

- ☐ Hotelogix;
- ☐ Oracle Hospitality;
- ☐ Frontdesk Anywhere.

Функції/Платформи	Frontdesk Anywhere	Hotelogix	Oracle Hospitality
Управління бронюваннями	+	+	+
Управління номерами	+	+	+
Управління рахунками	+	+	+
Управління персоналом	+	-	+
Управління запасами	-	+	+
Хмарна платформа	+	+	+
Інтуїтивний інтерфейс	+	+	+
Мобільні застосунки	+	+	-
Інтеграція з іншими системами	-	+	+
Вартість	-	+	-
Функціональність	-	-	+

4

Рисунок Б.4 - Слайд 4

Драйвери для з'єднання з базою даних

AZURE

- ❑ Azure ODBC Driver: Це драйвер, який надає можливість з'єднатися з базою даних Microsoft Azure за допомогою протоколу ODBC (Open Database Connectivity). Він дозволяє використовувати стандартний інтерфейс ODBC для взаємодії з базою даних;
- ❑ Azure SQL Database Connector: Це драйвер, спеціально розроблений для підключення до Azure SQL Database - послуги керування базами даних у хмарі Microsoft Azure. Він надає зручний інтерфейс для взаємодії з базою даних та виконання запитів;
- ❑ Azure Cosmos DB SDK: Це набір бібліотек та інструментів, які дозволяють з'єднатися з базою даних Azure Cosmos DB - глобально розподіленою, масштабованою базою даних у хмарі. Цей SDK надає функціональність для збереження, отримання та маніпулювання даними в Azure Cosmos DB.

5

Рисунок Б.5 - Слайд 5

Мови програмування

- ❑ C++;
- ❑ Python;
- ❑ Java;
- ❑ Ruby.

Мова програмування	Переваги	Недоліки
1 C++	2 - висока продуктивність і швидкодія; - контрольований доступ до ресурсів; - можливість оптимізації коду; - підтримка багатовитокowego програмування	3 - складна розробка та підтримка; - об'ємна екосистема готових рішень
Python	- простий та зрозумілий синтаксис; - багата екосистема готових рішень та бібліотек; - велика кількість доступної документації; - підтримка скриптового програмування	- висока продуктивність
Java	- переносність між платформами	- вищі витрати на ресурси
	- велика екосистема готових рішень та бібліотек; - підтримка багатовитокowego програмування; - сильна типізація та об'єктно-орієнтована парадигма	
Ruby	- лаконічний та зручний синтаксис; - багата екосистема готових рішень та фреймворків; - підтримка функціонального та об'єктно-орієнтованого програмування; - зручність розробки та підтримки коду	- висока продуктивність

6

Рисунок Б.6 - Слайд 6

Функціонал системи

- ☐ управління бронюваннями;
- ☐ управління номерами;
- ☐ управління рахунками;
- ☐ управління персоналом;
- ☐ управління запасами.

7

Рисунок Б.7- Слайд 7

Архітектура системи



Архітектура системи

8

Рисунок Б.8 - Слайд 8

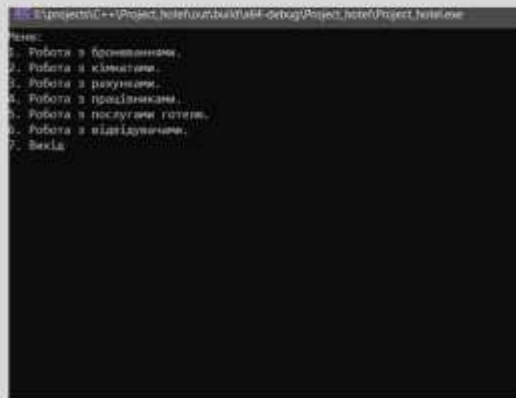
База даних



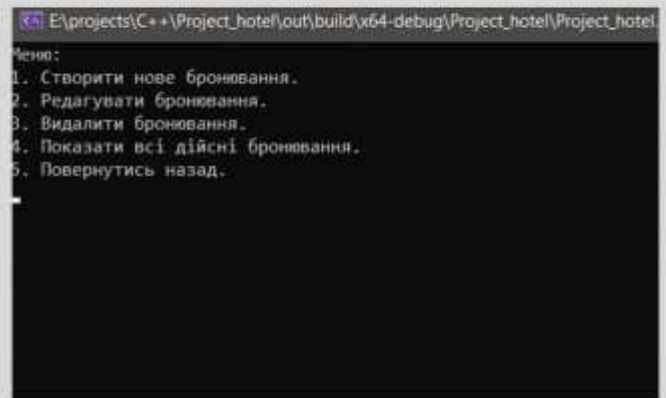
Схема бази даних

Рисунок Б.9 - Слайд 9

Інтерфейс застосунку



Меню застосунку



Меню функції робота з кімнатами

Рисунок Б.10 - Слайд 10

Сповіщення застосунку

```
E:\projects\C++\Project_hotel\out\build\x64-debug\Project_hotel\Project_hotel.exe
Створено нове бронювання:
ID бронювання: 14
Дата заїзду: 2023-06-06
Дата виїзду: 2023-06-07
ID кімнати: 2
ID гостя: 5
Ім'я гостя: Kostya
Для продовження натисніть будь-яку клавішу . . .
```

Сповіщення про створення нового бронювання

```
E:\projects\C++\Project_hotel\out\build\x64-debug\Project_hotel\Project_hotel.exe
Дата заїзду не може бути минулою.
Для продовження натисніть будь-яку клавішу . . .

E:\projects\C++\Project_hotel\out\build\x64-debug\Project_hotel\Project_hotel.exe
Кімната з ID 1 недоступна.
Для продовження натисніть будь-яку клавішу . . .
```

Обробка перевірок

11

Рисунок Б.11 - Слайд 11

Висновки

- ☐ проведено огляд і дослідження різних АСУ для підтримки готельного сервісу, виявлені їх переваги і недоліки;
- ☐ розглянуто традиційні підходи до підключення до баз даних;
- ☐ проведено аналіз традиційних хмарних платформ;
- ☐ також в роботі описано особливості використання системи;
- ☐ проведено відлагодження та тестування розробленого сервісу.

12

Рисунок Б.12 - Слайд 12