



Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота

Інформаційна система управління часовим розподілом ресурсів (за зразком Airbnb)

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

№	П.І.Б.	Група	Роль
1	Мохсені Софіані Микита Яхйяйович	КН-П-212	Розробник (backend)
2	Ковальов Данило Ярославович	КН-П-212	Розробник (frontend)
3	Шарпіло Дмитро Андрійович	КН-П-212	Розробник (backend)
4	Попов Петро Петрович	КН-П-212	Розробник (frontend)

Автор звіту: Мохсені Софіані Микита Яхйяйович

Ментор	Доцент кафедри ІТ та ФП, к.т.н., доцент Краковецький О.Ю.
--------	--

АНОТАЦІЯ

Робота була присвячена розробці функціонального веб-додатку з онлайн-бронюванням житла. Ця система спирається на вже існуючу систему [airbnb.com](https://www.airbnb.com), з покращеним інтерфейсом та функціональними можливостями.

Проект із серверної сторони була реалізована на ASP.NET Core 8.0 з використанням архітектури N-Tier Architecture, API-Centric Architecture, Domain-Driven Design, що дозволяє підтримувати модульність та незалежність функціональних частин проекту. Для роботи з даними були використані Microsoft SQL Server у сервісі Google Cloud SQL, EF Core та API, які реалізовані форматом RESTful API. Сервер для постійної роботи було опубліковано через сервіс Microsoft Azure, що дає змогу отримувати, відправляти дані, а також їх аналізувати. Додатково звернено увагу на аспекти безпеки проекту: авторизація та автентифікація реалізовані з використанням JWT Token, у той час як доступ до адміністративних функцій обмежений за допомогою ASP.NET Identity та ролей. Для забезпечення безпечного зберігання паролів користувачів їх шифрування виконується в базі даних за допомогою BCrypt.

З боку клієнтської частини проект розроблений з використанням бібліотеки React. Архітектура клієнтської частини побудована за feature-based принципом, при якому кожен функціональний модуль, такий як: авторизація, фільтрація, картки житла були ізольовані у власну директорію з компонентами, логікою стану, стилями та API-запитами. Для маршрутизації всередині програми використовується React Router DOM, що дозволяє реалізувати навігацію без перезавантаження сторінки. Для роботи зі станами у проект була інтегрована бібліотека Redux, дана бібліотека дозволяє позбавитися передачі станів між компонентами через “props”, а зберігати їх глобально і працювати з ними в будь-яких компонентах. Для роботи з отриманими даними, кешем та такими станами як (isLoading, isPending, isError, onSuccess) була інтегрована бібліотека React Query, яка також

дозволила абстрагуватися від локальних станів у компонентах. Для надсилання запитів на серверну частину проекту було додано бібліотеку Axios. Axios забезпечив більш гнучку та зручну роботу з HTTP запитамі порівняно з fetch. З використанням Axios було реалізовано функції взаємодії з Rest Api.

Результат проекту є масштабований веб-додаток із високим рівнем безпеки, сучасною архітектурою та гарною ефективністю, з можливістю подальшого розвитку.

ЗМІСТ

ВСТУП	5
Технічне завдання до проекту_____	6
РОЗДІЛ 1. Загальна характеристика системи	12
РОЗДІЛ 2. Реалізація клієнтська частини	15
РОЗДІЛ 3. ОЦІНКА	20
ВИСНОВКИ_____	25
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ	27

ВСТУП

На сьогоднішній день безліч речей, які потрібні для повсякденного життя, можна зробити все через пристрій з доступом в Інтернет, речі такі як: замовити їжу, знайти одяг, купити квиток в кіно і так далі. Інтернет дуже швидко розвивається та впроваджується в наші життя. І бронювання послуг є одним із цих завдань. На даний момент систем бронювання домінують кілька компаній та розробників програмного забезпечення, і є багато випадків, коли вони можуть бути корисними. Систему бронювання можна знайти скрізь: у кінотеатрах, лікарнях, салонах краси, ресторанах і так далі. І такі системи можуть бути складними, оскільки є багато різних варіацій структур та користувачів, що призводить до ускладнення розробки ресурсу.

Якщо система бронювання буде дуже гнучкою, то є високий шанс того, що вона може бути складною для користувача та адміністрації. Результатом може бути: непотрібні натискання з боку адміну, і при цьому включаються функції, в яких немає потреб. Системи для бронювання повинні бути простими та логічними у використанні одночасно для користувача та ринку.

У документі розглядаються проблеми, з якими ми зіткнулися під час розробки системи, а також описується повний цикл розробки цього проекту.

Звіт складається з трьох розділів, у яких описуються загальні характеристики системи, деталі створення клієнтської частини, а також результати її розгортання та тестування. Висновки відображають результати, досягнуті у цій частині проекту, загальний висновок - сукупність звітів усіх учасників проекту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

СИСТЕМА УПРАВЛІННЯ РЕЗЕРВУВАННЯМ РЕСУРСІВ
(за прикладом сервісу Airbnb.com)

Загальний опис.

Програмне забезпечення (ПЗ) призначене для створення розподіленої системи керування ресурсами, яке передбачає можливість проактивного управління, зокрема, резервування. Функції системи мають забезпечити повний життєвий цикл управління ресурсами: створення, актуалізація, використання та видалення (вживання). Головна особливість системи полягає у тому, що пріоритет використання обмежених ресурсів знаходиться у залежності від терміну його резервування.

Призначення:

ПЗ орієнтується на створення контейнеру управління обмеженими ресурсами на основі аналізу їх резервування. Оброблення конкурентних запитів на обмежені ресурси має враховувати показники пріоритетності, одним з яких є часовий показник моменту резервування. ПЗ створюється за прикладом сервісу бронювання обмеженої кількості місць, у той же час має складатись за принципом універсальності, мати змогу змінити цільове призначення.

Вимоги до функціоналу. Обов'язкова частина

Головна сторінка

Стартовий інтерфейс ПЗ (головна сторінка або активність) має за мету популяризацію системи. На ньому має бути забезпечено відображення помешкань, з можливістю вибирати категорії. Також головна сторінка має містити засоби авторизації та реєстрації або посилання на них.

Реєстрація користувачів:

Користувач ПЗ має змогу зареєструватись за допомогою авторизаційних даних (логіну/пароллю).

При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - електронна пошта. Засіб оперативного контактування при реєстрації підлягає верифікації через введення коду, пересланого через повідомлення довільного типу. Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.

Також передбачається наявність у системі користувачів з додатковими правами: адміністраторів. Реєстрація кореневого адміністратора системи має бути реалізоване при інсталяції системи, наступним користувачам змінюють права наявні адміністратори.

Авторизація та особистий кабінет користувача:

За фактом успішної авторизації користувачу надається доступ до особистого кабінету. Це окремий інформаційний ресурс (сторінка), що являє собою основне робоче місце користувача ПЗ. Повинен забезпечити усі необхідні засоби для:

- Пошук ресурсів за метаданими, групування та фільтрування результатів за різними ознаками, наявними у знайденому контенті. Для об'ємних результатів має бути забезпечена пагінація
- Створення замовлень на вільні ресурси
- Управління власними замовленнями: перегляд, пошук, підтвердження, відмова

Також засобами особистого кабінету має бути передбачено можливість зміни персональних даних.

Зворотній зв'язок

Користувач має змогу встановити власну оцінку для спожитого ресурсу. Також є можливість залишити коментар щоб інші користувачі могли більше дізнатись про ресурс.

Архітектура

ПЗ має бути спроектоване за розподіленою клієнт-серверною архітектурою з чітко відокремленими розподіленими вузлами системи:

Москир - візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.

Backend - серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.

Web App - додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів усіх категорій.

Усі модулі ПЗ (окрім візуальної моделі Москир) повинні обов'язково включати до свого складу засоби

- розгортання на новому пристрої (інсталяції) із встановленням початкових таблиць у БД та кореневого користувача-адміністратор

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контролю версій (вибір конкретної системи узгоджується групою проекту).

Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:

- HomeFU
 - home-fu-frontend-public
 - home-fu-backend-public

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

Backend - .NET технології із хмарним розміщенням (Azure), Google Cloud SQL

Web – JS, React, TS, HTML, Sass

Mockup - визначається ментором з дизайну

Команда проекту

Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проекту. Мінімальна команда проекту - два учасники, у такому складі один учасник виконує модуль, другий - програмну частину Web та розміткову частину Web.

Рекомендована команда проекту - чотири учасники, у такій команді два учасника відповідають за модуль Backend, інші два - програмну частину Web та розміткову частину Web.

Дозволяється включення до команди більшої кількості учасників.

Звітність

За результатами виконання завдань із створення ПЗ кожен з учасників команди проекту складає власний (індивідуальний) звіт, що складається з практичної частини та автореферату. Практична частина є прямим результатом роботи і подається у вигляді посилання на репозиторій (чи інший URL) або у вигляді архіву на довільному носіїві даних. Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU <http://www.gnu.org/prep/standards/>, у тому числі з додержанням вимог до коментування та документування коду.

Автореферат містить описову частину, у якій розкриваються основні концепти проекту, деталізуються завдання до виконавця, встановлюється актуальність роботи, її об'єкт та предмет, обґрунтовується вибір технологій та

відзначається шлях виконання та ступінь досягнення поставлених задач, посилення на джерела інформації.

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Архітектура платформи складається з таких частин:

«Frontend».

«Backend»,

«FRONTEND»

З основних аспектів веб-архітектури має бути єдиний стиль та інтерфейсна основа. Функціональна частина має бути реалізована з урахуванням єдиного стилю.

Проект повинен бути інтуїтивно зрозумілий і він повинен відповідати сучасним стандартам для користувальницького використання.

Наш проект має безліч сторінок для реалізації великого контенту, а також можливості додавання нових фіч, які дозволять поліпшити проект. Односторінковий проект, на даний момент є не актуальним і не має жодного сенсу, тому що додавання яких-небудь фіч, будуть обиратися в серверну частину, навантаженням на одній сторінці а також не можливості фізично додати будь-яку додаткову функціональність.

1. Проект повинен використовувати однакові елементи взаємодії з користувачем.
2. Розташування кожного елемента взаємодії повинні бути однакові
3. Розміри елементів повинні дотримуватися загального шаблону
4. Всі елементи на різних сторінках повинні дотримуватися загального стандарту, загального стилю.

5. Дизайн елементів має бути врахований та реалізований на початковій стадії розробки

Розробник Frontend повинен скласти грамотний план дій та реалізації, для виконання всіх умов і успішного реалізацію проекту. Від грамотного підходу розробника залежить реалізація самого проекту та його правильних підхід. У разі відхилення від норм або заздалегідь обговореного регламенту, розробка може бути більш тимчасово витратною або проявлятися конфлікти

«BACKEND»

Протягом життєвого циклу системи у «backend» зберігаються і обробляються дані, створенні та редагуванні у всій системі. Компоненти цього компонента включають проектну базу даних та API (Application Programming Interface), які дозволяють отримувати доступ до основних функцій системи через клієнську частину. Незамінна частина «backend» створена на мові програмування C# і Microsoft SQL Server через сервіс Google Cloud SQL у якості бази даних.

Онлайн-платформа має пропрієтарне ліцензію програмного забезпечення. Через це тільки правовласник має доступ до вихідного коду програми. Програмна частина платформи зберігається у приватному хмарному репозиторії веб-сервісу GitHub (через дотримання правил комерційної ліцензії) за посиланням: [GitHub](#). Доступ до готової користувальницької частини онлайн-платформи знаходиться за адресою: [посилання](#).

РОЗДІЛ 2

РЕАЛІЗАЦІЯ BACKEND

Для реалізації серверної частини було прийнято використовувати такі технології:

- Мова програмування: C#
- Фреймворк: ASP.NET Core 8.0 WebApi
- Середовище розробки: Visual Studio 2022
- Хост: Microsoft Azure
- База даних: Microsoft SQL
- Інструмент адміністрації БД: Google Cloud SQL
- Технології: EF Core
- Додаткові пакети: AutoMapper, BCrypt, MailKit, MimeKit, JwtToken
- API, де керують користувачами, паролями тощо: ASP.NET Identity

В основі серверної частини веб-програми лежить RESTful API. При написанні даного проекту використовувалися такі принципи проектування: Multi-tier / Layered Architecture, Repository Pattern, Unit of Work Pattern, Data Transfer Object Pattern, Dependency Injection, Service Layer Pattern. Така структура дає змогу масштабувати проект без нарощування труднощів.

Для зберігання та обробки інформації користувачів, їх місцезнаходжень та різних даних використовувалася база даних Microsoft SQL на платформі Google Cloud SQL для управління даними і моніторингу їх роботи БД. Для взаємозв'язку та опрацювання даних з бази даних використовувався об'єктно-реляційний відображувач (Object-Relational Mapper) разом з мовою запитів Language-Integrated Query (LINQ). Цей підхід сприяє швидкому і зрозумілому написанню коду.

У проекті було реалізовано надійну реєстрацію, авторизацію та аутентифікацію, дані користувача зберігаються в зашифрованому вигляді, що не дає можливості зловмисникам отримати доступ до облікового запису. При реєстрації користувач вказує пошту та пароль, після цього потрібно підтвердити пошту за допомогою коду, який буде надіслано на пошту, коли користувач підтвердить пошту, він зможе авторизуватися у свій обліковий запис, в іншому випадку, якщо перевірка буде не пройдена - обліковий запис буде недійсний доки не буде пройдено перевірку.

```
[HttpPost("register")]
0 references
public async Task<IActionResult> Register([FromBody] User user)
{
    if (await _context.Users.AnyAsync(u => u.Email == user.Email))
    {
        return BadRequest("Email уже используется");
    }

    // 1) При регистрации указывал роль "USER"
    user.Password = BCrypt.Net.BCrypt.HashPassword(user.Password);
    user.Role = "User"; // Присваиваем роль "User" по умолчанию

    // Генерируем код подтверждения
    user.EmailConfirmCode = GenerateConfirmationCode(); // Генерируем код

    _context.Users.Add(user);
    await _context.SaveChangesAsync();

    // Отправка письма с кодом подтверждения
    try{...}

    return Ok("Регистрация успешна. На вашу почту отправлен код подтверждения.");
}
```

У моменті авторизації користувача генерується токен, за допомогою якого всі запити будуть обробляти Bearer Token на дійсність та перегляду ролі. Якщо токен буде прострочено, користувачеві потрібно буде увійти до свого облікового запису для отримання нового токена.

```

[HttpPost("login")]
0 references
public async Task<IActionResult> Login([FromBody] UserLoginDto userLogin)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.Email == userLogin.Email);

    if (user == null || !BCrypt.Net.BCrypt.Verify(userLogin.Password, user.Password))
    {
        return Unauthorized("Неверный email или пароль");
    }

    // Проверяем, подтвержден ли email (код подтверждения должен быть пустым/null)
    if (!string.IsNullOrEmpty(user.EmailConfirmCode))
    {
        return StatusCode(403, "Пожалуйста, подтвердите ваш email, используя код, отправленный на вашу почту.");
    }

    var accessToken = GenerateJwtToken(user);
    var refreshToken = GenerateRefreshToken();
    var accessTokenExpiryTime = DateTime.UtcNow.AddMinutes(Convert.ToInt32(_configuration["Jwt:ExpirationMinutes"]));
    var refreshTokenExpiryTime = DateTime.UtcNow.AddDays(7);

    var tokenEntity = new Token(...);

    _context.Tokens.Add(tokenEntity);
    await _context.SaveChangesAsync();

    // 2) При авторизации возвращал роль юзера
    return Ok(new(...));
}

```

Щоб клієнтська частина не отримувала непотрібні дані з бази даних, були створені об'єкти, у яких записуються ті дані, які зберігають лише потрібну інформацію. Для цього були використані DTO (Data Transfer Object), в яких записується ті дані, на які чекає клієнтська частина. DTO є фундаментальним патерном у сучасному API-розробці дає ряд переваг, таких як: Поділ відповідальності, Чистий контракт API, Запобігання "Mass Assignment" / "Over-Posting" атак, Незалежність від БД, Версіонування API та Валідацію.

```

Data
├── DAL
│   ├── ContentDao.cs
│   ├── DataAccessor.cs
│   └── UserDao.cs
└── DTO
    ├── AmenityDto.cs
    ├── CardDetailDto.cs
    ├── CardDto.cs
    ├── CardResponseDto.cs
    ├── CardUpdateDto.cs
    ├── CategoryDto.cs
    ├── CommentResponseDto.cs
    ├── ConfirmEmailDto.cs
    ├── FiltersDto.cs
    ├── RatingDto.cs
    ├── ReservationDto.cs
    ├── ReviewCreateDto.cs
    ├── UserLoginDto.cs
    ├── UserPatchDto.cs
    ├── UserRatingCreateDto.cs
    └── UserRatingResponseDto.cs

```

Було ухвалено рішення зберігати дані про житла в такому форматі як Card та CardDetail для мінімізації навантаження на сервер при спробі отримати дані про всі доступні житла. При запиті на певне житло, крім даних із Card, підвантажується CardDetail, оскільки вона є деталізацією батьківської сутності, а не незалежною. Для їхнього зв'язку був використаний Shared Primary Key - такий підхід дає плюси, такі як: Гарантія відносин 1:1, Спрощення об'єктної моделі в ORM, Семантична ясність та Оптимізація зберігання.

```
namespace HomeFuBack.Models.Housing
{
    11 references
    public class CardDetail
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)] // Id не автогенерируется
        12 references
        public int Id { get; set; }

        // (Shared Primary Key)
        [ForeignKey("Id")]
        28 references
        public Card Card { get; set; } = null!;

        // Общая информация
        8 references
        public int NumberOfGuests { get; set; }
        3 references
        public int NumberOfBedrooms { get; set; }
        3 references
        public int NumberOfBeds { get; set; }
        3 references
    }
}
```

Як писалося раніше, для підтвердження пошти потрібно ввести код, який надходить на електронну пошту. Для реалізації надсилання листів були використані MimeKit та MailKit, за допомогою яких крім підтвердження реєстрації облікового запису можна підготувати сервіс на відновлення пароля, повідомлення про дії, маркетингові та інформаційні розсилки.

```

namespace HomeFuBack.Helpers
{
    2 references
    public class EmailSender : IEmailSender
    {
        private readonly EmailSettings _smtpSettings;

        0 references
        public EmailSender(IOptions<EmailSettings> smtpSettings)
        {
            _smtpSettings = smtpSettings.Value;
        }

        4 references
        public async Task SendEmailAsync(string toEmail, string subject, string message)
        {
            var email = new MimeMessage();
            email.From.Add(new MailboxAddress(_smtpSettings.SenderName, _smtpSettings.SenderEmail));
            email.To.Add(MailboxAddress.Parse(toEmail));
            email.Subject = subject;
            email.Body = new TextPart(MimeKit.Text.TextFormat.Html) { Text = message };

            using (var client = new SmtplibClient())
            {
                await client.ConnectAsync(_smtpSettings.SmtpServer, _smtpSettings.SmtpPort, SecureSocketOptions.StartTls);
                await client.AuthenticateAsync(_smtpSettings.Username, _smtpSettings.Password);
                await client.SendAsync(email);
                await client.DisconnectAsync(true);
            }
        }
    }
}

```

У проєкті також для модульної та чистої структури були створені Helper класи, які використовуються для інкапсуляції логіки, яка буде повторно використовуватись. У проєкті є такі класи, як UniqueFileName, використовується для генерації унікальних імен файлів при завантаженні та запобігає колізії імен при завантаженні декількох файлів з однаковим ім'ям.

```

2 references
public class UniqueFileName
{
    2 references
    public static string GetUniqueFileName(string fileName)
    {
        fileName = Path.GetFileName(fileName);
        return $"{Guid.NewGuid()}_{Path.GetExtension(fileName)}";
    }
}

```

Клас IEmailSender та EmailSender – перший клас це інтерфейс для відправки листів поштою, а другий клас – його реалізація. Цей клас має користь у цьому, що абстрагуємося від конкретної реалізації відправлення листів і централізація - що стосується відправлення листів, зосереджено щодо одного методі. Клас ClaimsPrincipalExtensions потрібен для вилучення конкретних claims з ClaimsPrincipal.

```

0 references
public static class ClaimsPrincipalExtensions
{
    3 references
    public static Guid GetUserId(this ClaimsPrincipal principal)
    {
        if (principal == null)
            throw new ArgumentNullException(nameof(principal));

        var userIdClaim = principal.FindFirst(ClaimTypes.NameIdentifier);
        if (userIdClaim == null)
        {
            userIdClaim = principal.FindFirst("sub"); // JwtRegisteredClaimNames.Sub
        }

        if (userIdClaim != null && Guid.TryParse(userIdClaim.Value, out Guid userId))
        {
            return userId;
        }

        throw new InvalidOperationException("User ID claim not found or not in GUID format.");
    }
}

```

У проєкті для кожного об'єкта реалізований CRUD, так як це базовий набір для операцій роботи з постійним сховищем даних, і частина функцій має розмежування прав доступу – наприклад: користувач може побачити тільки свої резервації, а адмін – всі резервації. Приклад коду де саме стоїть обмеження

```

[HttpGet]
[Authorize(Roles = "Admin")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
0 references
public async Task<ActionResult<IEnumerable<ReservationResponseDto>>> GetAllReservations()
{
    var reservations = await _context.Reservations
        .Include(r => r.Card)
        .Include(r => r.User)
        .ToListAsync();

    var reservationDtos = reservations.Select(r => MapToReservationResponseDto(r)).ToList();
    return Ok(reservationDtos);
}

[HttpGet("{id}")]
[Authorize]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
1 reference
public async Task<ActionResult<ReservationResponseDto>> GetReservation(int id)
{
    var reservation = await _context.Reservations
        .Include(r => r.Card)
        .Include(r => r.User)
        .FirstOrDefaultAsync(r => r.Id == id);

    if (reservation == null)
    {
        return NotFound($"Резервація з ID {id} не знайдена.");
    }
}

```

Також був застосований шаблон проектування веб-додатків та баз даних - Soft Delete. Такий шаблон дозволяє обійтися без складних каскадних видалень, тобто підтримувати цілісність посилань, у користувача або адміна буде можливість відновити "видалені" дані

```
[DataType(DataType.Currency)]
[Column(TypeName = "decimal(18, 2)")]
11 references
public decimal Price { get; set; }

14 references
public CardDetail? CardDetail { get; set; } // Card

15 references
public bool IsDeleted { get; set; }

2 references
public Card()
{
    CardCategories = [];
    ImageUrls = [];
    IsDeleted = false;
}
```

```
var card = cardDetail.Card;
if (card == null)
{
    return NotFound($"Связанная основная карточка для Card");
}

card.IsDeleted = true;
_context.Entry(card).State = EntityState.Modified;

try
{
    await _context.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException)
{
}
```

РОЗДІЛ 3

ОЦІНКА

Після проектування архітектури проекту та розробки головних компонентів повинні бути розглянуті вимоги сайту. Основою оцінювання вимог спиратиметься на вимоги користувачів з метою майбутнє.

Основи

Платформа AirBnb.com була взята у приклад для створення платформи. Зокрема, дизайн та функціональність повинні бути схожі на AirBnb.com, з кількома покращеннями та виправленнями. Беручи це до уваги у контексті архітектури додатків, є багато рішень, які слід розглянути.

Робота була спрямована на створення ефективної, безпечної та масштабованої API-структури. Дана архітектура повинна працювати на

основі компонентів так, щоб фронтенд частина могла звертатися до API-енд поінтів для отримання та відправлення даних. Як писалося раніше, проект був побудований на RESTful API, і дана архітектура дає можливість різними блоками незалежно взаємодіяти з даними, наприклад: профіль користувача, бронювання тощо. Також було введено систему аутентифікації та авторизації на основі JWT токенів, за допомогою яких можна обмежити несанкціонований доступ до ресурсів. Деякі блоки функцій контролерів мають перевірку на авторизацію, а також перевірку ролі користувача для обмеження доступу до адмін-функцій. Ця система відповідає вимогам адаптацій до інформаційних вимог користувачів. Також використано принцип поділів відповідальності кожного контролера та сервісів, що давало можливість паралельної розробки з фронтенд частиною.

Масштабованість

Масштабованість проекту також був важливим для бекенд частини, так як це дозволяє ефективно зростати і справлятися з зростаючими вимогами ринку, так і користувачів, забезпечуючи високу надійність та продуктивність веб-додатку. Це комплексний підхід до проектування, що якщо при неправильній розробці основної архітектури проекту, могла б привести низку проблем при додаванні нових компонентів, що призвело б до втрати часу та ресурсів. Для того, щоб не зіткнутися з цими проблемами, потрібно продумати хорошу архітектуру, реалізувати незалежні компоненти та уникати проблем із залежностями. Для цього були використані такі архітектурні патерни, як: Модульний дизайн - модульний підхід, де окремі сервіси є незалежними. Такий підхід дає можливість додавати нові функції і при цьому не впливати вже на існуючі. Ефективне використання ORM дозволяє легко масштабувати веб-додаток. Основні зміни управляється централізовано, що важливо для додавань кількостей компонентів у подальшому.

Можливі майбутні вдосконалення веб-застосунку

Під час роботи стикалися з чималою кількістю проблем, які на протязі часу були вирішені та досягли успішної реалізації базової функціональності бекенд частини проекту, все ж є кілька напрямків, де варто покращити та додати у майбутньому. Спочатку потрібно оптимізувати продуктивність та кешування для того, щоб знизити навантаження на БД, планується впровадити механізм кешування Redis у бекенд, це дозволить отримувати дані повторно на високому рівні. Для подальшої роботи з масштабуванням потрібно дотримуватись мікросервісної архітектури для поділу компонентів на дрібніші частини, це дозволить надалі зробити розробку простішою та зрозумілою. Як говорилося раніше, хоч і проблеми з якими ми зіткнулися були вирішені та компоненти оновлені з використанням різних патернів та архітектур, що покращують можливість масштабування, напевно, залишиться кілька технічних проблем, які ще не були зафіксовані і їх варто виправити.

ВИСНОВКИ

Цей проект є на етапі ранньої розробки на момент написання документа і мають деякі складові частини які необхідно додати та вдосконалити. Вся робота з цим проектом описана в даному документі та у розділі “Оцінка” зазначено багато вже виконаною роботи та залишилося ще багато завдань для покращення цього проекту та додавання функцій для якості життя (Quality of Life). Я припускаю що всі недоліки цього проекту та відсутні елементи будуть вирішені у найближчому майбутньому.

На основі нашого досвіду у проектуванні архітектури можна зробити висновки про те, що необхідно починати роботу над проектуванням до початку інтенсивних фаз реалізацій. Основними складнощами під час виконання проекту є правильне структурування об'єктів та оптимізація часових рамок через значний витрати часу на виправлення помилок і аналіз проблем. Наприклад при реалізації CRUD в одного об'єкта, яким був погано реалізований зв'язок з іншими об'єктами, привела проблему з його видаленням - при спробі видалити даний об'єкт з БД, вона викликала помилку, що призводила до видалення всіх даних його таблиці і також пов'язаних.

Але в результаті, при виникненні помилок, було виділено чимало сил на вивчення матеріалу для запобігання цим проблемам, що призвело до того, що цей проект має вже хорошу архітектуру і структуру, що має хороший потенціал для розширення можливостей веб-додатку. Однією з важливих переваг є використання ORM, що дозволяє розширити кількість об'єктів, не впливаючи на вже існуючі, які зберігають у собі дані, та поділ проекту на різні частини незалежних сервісів, що також дає змогу додати інші послуги, без зміни та впливу на старих. І для тривалої підтримки проекту були обрані популярні фреймворки, які дотримуються постійних оновлень для покращення продуктивності та функціональності.

У момент написання документа, що розпочалася у березні 2025 року, було чимало падінь, але й також злетів, у зв'язку з чим швидкість підготовки проекту сильно змінювалась. Найчастіше це виникало через обмеження ресурсів для хосту бекенд проекту та бази даних, ситуаціями в житті кожного члена команди та ситуацій країни. І для того, щоб дотримуватися високої швидкості та якості проекту, у разі потреби член команди міг делегувати завдання іншим, що призвело до швидкої підготовки базової структури та архітектури проекту.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ
КЛІЄНТСЬКА ЧАСТИНИ

№ п.п.	Ресурс
1.	C#; Документація: посилання .
2.	Visual Studio 2022; Документація: посилання .
3.	.NET Core; Документація: посилання .
4.	ASP.NET Core; Документація: посилання .

5.	Entity Framework Core; Документація: посилання .
7.	MailKit ; Документація: посилання .
8.	AutoMapper; Документація: посилання .
9.	Microsoft SQL; Документація: посилання .
10.	Google Cloud SQL; Документація: посилання .
11.	Microsoft Azure; Документація: посилання .
12.	Google Gemini; Документація: посилання .
13.	GitHub; Документація: посилання .