



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: Шарпіло Дмитро Андрійович

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

АНОТАЦІЯ

Робота була посвящена розробці функціонального веб-приложения с онлайн-бронированием жилья. Данная система опирается на уже существующую систему [airbnb.com](https://www.airbnb.com), с улучшенным интерфейсом и функциональных возможностей.

Проект с серверной стороны была реализована на ASP.NET Core 8.0 с использованием архитектур N-Tier Architecture, API-Centric Architecture, Domain-Driven Design, что позволяет поддерживать модульность и независимость функциональных частей проекта. Для работы с данными были использованы Microsoft SQL Server в сервисе Google Cloud SQL, EF Core и API которые реализованы форматом RESTful API. Сервер для постоянной работы была опубликована через сервис Microsoft Azure, что позволяет получать, отправлять данные, а также их анализировать. Также было выделено внимание безопасности проекта: авторизация и аутентификация сделаны на базе JWT Token, а доступ до административной функциональности была ограничена через ASP.NET Identity и роли. Для безопасного хранения пароля пользователя шифруются в базе данных с помощью BCrypt.

Со стороны клиентской части, проект разработан с использованием библиотеки React. Архитектура клиентской части построена по feature-based принципу, при котором каждый функциональный модуль, такой как: авторизация, фильтрация, карточки жилья были изолированы в собственную директорию с компонентами, логикой состояния, стилями и API-запросами. Для маршрутизации внутри приложения используется React Router DOM, позволяющий реализовать навигацию без перезагрузки страницы. Для работы с состояниями в проект была интегрирована библиотека Redux, данная библиотека позволяет избавиться от передачи состояний между компонентами через “props”, а хранить их глобально и работать с ними в любых компонентах. Для работы с полученными данными, кешем и такими

состояниями как (isLoading, isPending, isError, onSuccess) была интегрирована библиотека React Query, которая также позволила абстрагироваться от локальных состояний в компонентах. Для отправки запросов на серверную часть проекта была добавлена библиотека Axios. Axios обеспечил более гибкую и удобную работу с HTTP запросами по сравнению с fetch. С использованием Axios были реализованы функции для взаимодействия с Rest Api.

Результат проекта есть масштабируемый веб-приложение с высоким уровнем безопасности, современной архитектурой и хорошей эффективностью, с возможностью дальнейшего развития.

ЗМІСТ

ВСТУП	5
Технічне завдання до проекту	6
РОЗДІЛ 1. Загальна характеристика системи	10
РОЗДІЛ 2. Реалізація клієнтська частини	13
РОЗДІЛ 3. ОЦІНКА	21
ВИСНОВКИ	<u>23</u>
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ	24

ВСТУП

На сегодняшний день множество вещей, которые нужны для повседневной жизни можно сделать все через устройство с доступом в Интернет, вещи такие как: заказать еду, найти одежду, купить билет в кино и так далее. Интернет очень быстро развивается и внедряется в наши жизни. И бронирование услуг является одним из этих задач. На данный момент систем бронирования доминируют несколько компаний и разработчиков программного обеспечения, и есть много случаев, когда правда они могут быть полезными. Систему бронирования можно найти везде: в кинотеатрах, в больницах, в салонах красоты, в ресторанах и так далее. И такие системы могут быть сложными, так как есть много разных вариаций структур и пользователей, что приводит к усложнению разработки ресурса.

Если система бронирования будет очень гибкой, то есть высокий шанс того, что она может быть сложной для пользователя и администрации. Результатом может быть: ненужные нажатия со стороны админа, и при этом включаются те функции, в которых нет потребностей. Системы для бронирования должны быть простыми и логичными в использовании одновременно для пользователя и рынка.

В документе рассматриваются проблемы, с которыми мы столкнулись во время разработки системы, а также описывается полный цикл разработки данного проекта.

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

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

Рекомендована команда проєкту - 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 проєкту 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 проєкту, 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 проекту були потрібні такі елементи:

1. Вимоги до реалізації проекту
2. Інструменти, що використовувалися для реалізації проекту
3. Визначення архітектури проекту
4. Реалізація коду

Вимоги до реалізації проекту

Для бекенд-частини нашого проекту потрібно було зробити перші кроки, беручи до уваги можливі зміни та масштабованість системи. Попередньо ми погодили фреймворки та мови програмування, щоб не було конфліктів у коді.

Фільтрація та бронювання по суті основний функціонал проекту. Для легкого додавання чи зміни функцій, а також для масштабування, архітектура була зроблена гнучкою, через що немає потреби повністю переробляти базову логіку. Крім того, це дозволить легше інтегрувати майбутні оновлення.

API фільтрації реалізований так, що дозволив фронтенду користуватися всіма параметрами (дати, кількість гостей) для точного пошуку та частковими запитамі (тільки по локації або рядку пошуку). Тому потрібно адаптувати логіку використання фільтрів, щоб користувач міг не вводити всі дані для безпосереднього пошуку, якщо вони не потрібні.

Між бекендом і фронтендом було потрібно зробити контроль цілісності даних DTO (Data Transfer Object) для того, щоб результати резервацій та параметри фільтрів здійснювалися без помилок.

Для надійності було важливо гарантувати точність фільтра під час пошуку

житла. Необхідно перевіряти на перетин дат для резервації, задіяти всі критерії фільтрації (локацій, кількості відвідувачів, періоду дат). Також потрібно запобігати подвійному бронюванню однієї і тієї ж дати.

Все, що пов'язано з резервацією, вимагає авторизацію та автентифікацію користувача. Обмеження доступу до даних резервації тільки для їхнього власника або адміна, а дані користувача та житла надійно захищені.

Інструменти, що використовувалися для реалізації проекту

Для функціоналу фільтрації та резервації житла використовувався C# як основна мова програмування, і .Net Core як фреймворк, тому що ми маємо з ними досвід і у зв'язку з тим, що час обмежений, недоцільно використовувати щось інше. Для зв'язку з базою даних використовувався EF Core. Він спростив роботу з даними, надавши можливість виконувати важкі запити на фільтрацію (перевірка перетину дат резервацій) за допомогою LINQ-запитів.

Середовище розробки: Visual Studio 2022.

Postman – використовувався для тестів API-ендпоінтів: відправки http-запитів з параметрами та перевіркою відповідей сервера.

Визначення архітектури проекту

Архітектура у нас це N-Tier Architecture, тобто багатоварштовий підхід, він нам забезпечує гнучкість і масштабованість фільтрацій та резервацій.

- API(контролери): приймаємо http-запити, робить валідацію вхідних даних за допомогою DTO та застосовує авторизацію/автентифікацію. Також формує відповіді клієнту.
- Сервіси: основні правила та логіка роботи з фільтрами та резерваціями. Валідація "бізнес-правил" (наприклад, перевірка перетинів дат або

розрахунок місткості).

- Доступ до даних (DbContext, Моделі): зв'язок із базою даних через EF Core, визначає структуру даних (наприклад моделі: Reservation, Card, User).
- DTOs: необхідні для передачі даних між шарами програми та клієнтом. Спрощують валідацію і дають контроль над вхідними та вихідними даними API.

Реалізація коду

Основна робота була на створення ефективних інструментів для фільтрації житла та управління резервацією.

ReservationController

Для роботи з резерваціями було створено ReservationController. У ньому знаходяться API-ендпоінти для наших операцій: створення, отримання, зміни та скасування броні.

- POST(створення резервації):
- Коли ми створюємо нову резервацію, система робить сувору валідацію даних (точність і відсутність у минулому) і дивиться щоб житло дотримувалося зазначеної місткості. Важливим моментом є перевірка на перетин дат. А саме триває перевірка з уже існуючими бронюваннями, що усуває можливість подвійного резервування.

```

[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status409Conflict)]

public async Task<ActionResult<ReservationResponseDto>> PostReservation([FromBody] ReservationDto reservationDto)
{
    if (!ModelState.IsValid) return BadRequest(ModelState);
    if (reservationDto.CheckInDate >= reservationDto.CheckOutDate) return BadRequest("Дата выезда должна быть после даты заезда.");

    var card = await _context.Cards.Include(c => c.CardDetail).ThenInclude(cd => cd.Host).FirstOrDefaultAsync(c => c.Id == reservationDto.CardId);
    if (card == null) return NotFound("Жилье не найдено.");

    var totalGuests = reservationDto.Adults + reservationDto.Children;
    if (totalGuests > card.CardDetail.NumberOfGuests) return BadRequest($"Количество гостей ({totalGuests}) превышает допустимое.");

    var hasOverlap = await _context.Reservations.AnyAsync(r =>
        r.CardId == reservationDto.CardId && r.Status != ReservationStatus.Cancelled && r.Status != ReservationStatus.Completed &&
        (reservationDto.CheckInDate < r.CheckOutDate && reservationDto.CheckOutDate > r.CheckInDate));
    if (hasOverlap) return Conflict("Выбранные даты недоступны. Есть пересечения с существующими резервациями.");
}

```

Після цього бронь зберігається в базі даних (зі статусом "Confirmed"), і надсилає email-повідомлення клієнту та власнику житла.

```

await _context.Entry(reservation).Reference(r => r.User).LoadAsync(); // Данных клиента для использования в письме

try
{
    var client = reservation.User;
    var host = card.CardDetail.Host;

    var numberOfNights = (reservation.CheckOutDate - reservation.CheckInDate).Days;
    var totalPrice = numberOfNights * card.Price;

    //Письмо для клиента
    var clientSubject = $"Ваша резервация в {card.Name} подтверждена!";
    var clientMessage = $"
    <h1>Резервация успешна!</h1>
    <p>Здравствуйте, {client.FirstName},</p>
    <p>Ваша резервация для жилья ""{card.Name}"" была успешно создана и подтверждена.</p>
    <h3>Детали вашей поездки:</h3>
    <ul>
    <li><strong>Дата заезда:</strong> {reservation.CheckInDate:dd MMMM yyyy}</li>
    <li><strong>Дата выезда:</strong> {reservation.CheckOutDate:dd MMMM yyyy}</li>
    <li><strong>Количество ночей:</strong> {numberOfNights}</li>
    <li><strong>Гости:</strong> {reservation.Adults} взрослых, {reservation.Children} детей, {reservation.Infants} младенцев</li>
    <li><strong>Питомцы:</strong> {reservation.Pets}</li>
    <li><strong>Итоговая сумма:</strong> {totalPrice:C}</li>
    </ul>
    <p>Спасибо, что выбрали HomeFu!</p>";

    await _emailSender.SendEmailAsync(client.Email, clientSubject, clientMessage); // Данных хоста для использования в письме
}

```

```

// Письмо для хоста
if (host.Email != client.Email)
{
    var hostSubject = $"Новая резервация вашего жилья: {card.Name}";
    var hostMessage = $"
    <h1>У вас новая резервация!</h1>
    <p>Здравствуйте, {host.FirstName},</p>
    <p>Ваше жилье ""{card.Name}"" было зарезервировано.</p>
    <h3>Детали резервации:</h3>
    <ul>
    <li><strong>Имя гостя:</strong> {client.FirstName} {client.LastName}</li>
    <li><strong>Email гостя:</strong> {client.Email}</li>
    <li><strong>Дата заезда:</strong> {reservation.CheckInDate:dd MMMM yyyy}</li>
    <li><strong>Дата выезда:</strong> {reservation.CheckOutDate:dd MMMM yyyy}</li>
    <li><strong>Гости:</strong> {reservation.Adults} взрослых, {reservation.Children} детей, {reservation.Infants} младенцев</li>
    <li><strong>Питомцы:</strong> {reservation.Pets}</li>
    <li><strong>Итоговая сумма:</strong> {totalPrice:C}</li>
    </ul>
    <p>Вы можете управлять резервациями в вашей панели управления HomeFu.</p>";

    await _emailSender.SendEmailAsync(host.Email, hostSubject, hostMessage);
}

catch (Exception ex)
{
    // Логируем ошибку, но не прерываем основной процесс.
    Console.WriteLine($"[WARNING] Ошибка при отправке email-уведомлений для резервации ID:{reservation.Id}. Ошибка: {ex.Message}");
}

```

- GET(Отримання резервацій):

Інформація конкретної резервації ми отримуємо за допомогою ID. Система автоматично витягує пов'язані дані про житло (Card) та користувача (User). Також важливим аспектом був контроль доступу. Простий користувач може переглядати тільки свої резервації. А “адміни” бачать усі резервації, які є у системі. Дізнаємося ми активного користувача та його роль за допомогою UserID.

```
[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($"Резервация с ID {id} не найдена.");
    }

    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (!User.IsInRole("Admin") && reservation.UserId.ToString() != currentUserId)
    {
        return Forbid();
    }

    return Ok(MapToReservationResponseDto(reservation));
}
```

- POST/cancel

Користувачі (і адміни) можуть скасувати свою резервацію, змінюючи її на статус “Cancelled”. Йде перевірка, щоб броню вже не було скасовано або завершено.

```

[HttpPost("{id}/cancel")]
[Authorize]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 references
public async Task<IActionResult> CancelReservation(int id)
{
    var reservation = await _context.Reservations.FindAsync(id);

    if (reservation == null) {return NotFound($"Резервация с ID {id} не найдена.");}

    var userIdClaim = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    if (!User.IsInRole("Admin") && (string.IsNullOrEmpty(userIdClaim)
        || !Guid.TryParse(userIdClaim, out Guid userIdGuid)
        || reservation.UserId != userIdGuid))
    {
        return Forbid("Вы можете отменять только свои резервации.");
    }

    if (reservation.Status == ReservationStatus.Cancelled || reservation.Status == ReservationStatus.Completed)
    {
        return BadRequest("Невозможно отменить резервацию с текущим статусом.");
    }

    reservation.Status = ReservationStatus.Cancelled;
    await _context.SaveChangesAsync();

    return NoContent();
}

```

- DELETE

Ще було зроблено ендпоінт для видалення резервацій з бази даних. Операція доступна тільки "Адмінам", це допомагає уникнути випадкових чи неавторизованих видалень бронювань. Перевірка на наявність тієї чи іншої резервації здійснюється за ID.

```

[HttpDelete("{id}")]
[Authorize(Roles = "Admin")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status404NotFound)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
0 references
public async Task<IActionResult> DeleteReservation(int id)
{
    var reservation = await _context.Reservations.FindAsync(id);
    if (reservation == null)
    {
        return NotFound($"Резервация с ID {id} не найдена.");
    }

    _context.Reservations.Remove(reservation);
    await _context.SaveChangesAsync();

    return NoContent();
}

```

FiltersController

Контролер фільтрації дає нам можливість шукати житло.

- В API у нас реалізована адаптивна логіка фільтрації: якщо вказати дату та кількість гостей, то буде пошук за доступністю, який використовує суворі фільтри за датами та місткістю. В іншому випадку буде “загальний пошук” (за локаціями, пошуковим рядком), який поверне нам усі відповідні картки.

Метод `FilterByAvailability`, вибирає логіку та ініціалізація запиту:

```
[HttpGet("availability")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public async Task<ActionResult<IEnumerable<CardResponseDto>>> FilterByAvailability([FromQuery] FiltersDto filterDto)
{
    // Валидация DTO
    if (!ModelState.IsValid) return BadRequest(ModelState);

    // Определяем режим фильтрации: полная проверка доступности (с датами/гостями) или общая фильтрация (только локация/поиск)
    bool isFullAvailabilityCheck = filterDto.CheckInDate.HasValue || filterDto.CheckOutDate.HasValue ||
        filterDto.Adults.HasValue || filterDto.Children.HasValue ||
        filterDto.Infants.HasValue || filterDto.Pets.HasValue;

    // Инициализируем базовый LINQ-запрос к карточкам, загружая необходимые связанные данные
    var query = _context.Cards
        .Include(c => c.CardDetail)
        .Include(c => c.Location)
        .Include(c => c.CardCategories)
        .ThenInclude(cc => cc.Category)
        .Where(c => !c.IsDeleted)
        .AsQueryable();
```

Фільтр по локації та пошуковому рядку:

```
// Фильтрация по ID Локации
if (filterDto.LocationId.HasValue && filterDto.LocationId.Value > 0)
{
    query = query.Where(c => c.LocationId == filterDto.LocationId.Value);
}

// Фильтрация по поисковому запросу (SearchTerm)
if (!string.IsNullOrEmpty(filterDto.SearchTerm))
{
    var searchTerm = filterDto.SearchTerm.Trim().ToLower();
    query = query.Where(c => (c.Name != null && c.Name.ToLower().Contains(searchTerm)) || // Поиск по имени жилья
        (c.Location != null && c.Location.Name.ToLower().Contains(searchTerm))); // Поиск по имени локации
}
```

Суворі фільтри (дати, кількість гостей, перетину резервацій) - тільки для "пошуку за доступністю"

Підготовка дат для запиту:

```
// 1. Подготовка дат для запроса (только если это полная проверка доступности)
DateTime checkIn = DateTime.MinValue; // Инициализируем минимальными значениями
DateTime checkOut = DateTime.MinValue;

if (isFullAvailabilityCheck)
{
    checkIn = filterDto.CheckInDate?.Date ?? DateTime.UtcNow.Date;
    checkOut = filterDto.CheckOutDate?.Date ?? checkIn.AddYears(1); // Большая дефолтная дата

    // Валидация дат для полной проверки
    if (checkIn >= checkOut)
    {
        return BadRequest("Дата выезда должна быть после даты заезда.");
    }
    if (checkIn < DateTime.UtcNow.Date)
    {
        return BadRequest("Дата заезда не может быть в прошлом.");
    }
}
```

Розрахунок загальної кількості відвідувачів для місткості

```
int totalGuestsForCapacity = 0; // Инициализируем 0, чтобы не влияло на базовый запрос

if (isFullAvailabilityCheck)
{
    // Проверка, что есть хотя бы 1 взрослый
    if (filterDto.Adults.GetValueOrDefault(0) == 0)
    {
        return BadRequest("Для бронирования необходимо указать хотя бы 1 взрослого.");
    }

    // Рассчитываем общую вместимость, включая всех гостей
    totalGuestsForCapacity = filterDto.Adults.Value + // Используем .Value, т.к. уже проверили, что Adults >= 1
        filterDto.Children.GetValueOrDefault(0) +
        filterDto.Infants.GetValueOrDefault(0); // Младенцы тоже учитываются в capacity

    if (totalGuestsForCapacity == 0 && filterDto.Pets.GetValueOrDefault(0) == 0) // Если все 0, включая питомцев
    {
        return BadRequest("Необходимо указать хотя бы 1 гостя (взрослого, ребенка или младенца), если нет питомцев.");
    }
}
```

Фільтрування по місткості житла

```
if (isFullAvailabilityCheck)
{
    query = query.Where(c =>
        c.CardDetail != null &&
        c.CardDetail.NumberOfGuests >= totalGuestsForCapacity
    );
}
```

Головна фільтрація Тут ми виключаємо картки у яких немає активної резервації, що перетинається з обраним періодом заїзду/виїзду.

```
if (isFullAvailabilityCheck)
{
    query = query.Where(card => !_context.Reservations.Any(reservation =>
        reservation.CardId == card.Id &&
        reservation.Status != ReservationStatus.Cancelled &&
        reservation.Status != ReservationStatus.Completed &&
        (checkIn < reservation.CheckOutDate && checkOut > reservation.CheckInDate)
    ));
}
```

РОЗДІЛ 3

ОЦІНКА

Оцінка веб-сайту - це досконалий аналіз структурного проекту і ключових компонентів реалізованих у ньому.

Основи

Завданням було створити платформу, яка ідейно та функціонально буде схожа на сервіс Airbnb. В основі лежить Rest API. Це дало нам чіткий та масштабований спосіб зв'язку між бекендом та фронтендом, що буде важливим моментом для фільтрації та резервації. Слід зазначити, що зразки з оригіналу значно змінилися, що з нашої думки створило унікальний підсумковий проект.

Працюючи над фільтрацією і резервацією треба було сфокусувати увагу до чітких контролерів і моделей даних, що дало нам надійну роботу функціоналу. Авторизація та аутентифікація дали можливість обмежити або розширити здатність користувача до взаємодії з API-ендпоінтами (наприклад, управління лише своїми резерваціями).

Масштабованість

Масштабованість проекту нам потрібна, щоб уникнути проблем на бекенді якщо додаватимемо новий функціонал. У контексті фільтрації та резервації ці проблеми можуть виникнути при невдалих запитах або помилках, які пропустили в ранніх етапах. Вони можуть призвести до збоїв системи при високих навантаженнях.

Компоненти бекенда зроблені з упором на їхню незалежність. Отже, якщо впровадити нові функції (наприклад, нові умови фільтрації), вони не будуть негативно впливати на існуючі та викликати помилки.

Також було використано ORM (Entity Framework Core). Воно абстрагує взаємозв'язок із базою даних, спрощуючи управління схемою бд, написання та оптимізація запитів.

Можливі майбутні вдосконалення веб-застосунку

Незважаючи на всі проблеми, ми змогли реалізувати головний функціонал. Але також існують напрями для подальшого розвитку та покращення серверної частини, що дозволило б розширити можливості.

Одним із вдосконалень є модальне вікно фільтру. У цьому місці у нас буде діапазон цін, зручності, такі як Wi-Fi або кондиціонер, та варіанти бронювання. У міру зростання кількості об'єктів нерухомості та бронювань, важкі запити фільтрів надаватимуть велике навантаження на сервер. Рішенням буде покращення запитів, налаштування більш сучасних систем кешування на бекенді і, можливо, перехід на більш ефективну базу даних.

Також, локалізація та підтримка інших мов може бути наступною. Оскільки ми хочемо охопити весь світ, ми повинні бути готовими показувати дані різними мовами. Потрібно буде зберігати назви локацій, категорій житла та тексту

підтверджень резервацій різними мовами, щоб здійснити локалізацію всім користувачів.

Також критично важливою є інтеграція з надійними платіжними шлюзами для нашого бронювання. Наразі резервація може фіксувати бронь, проте ніяк не обробляє фінансові транзакції. У майбутньому потрібно буде зробити API-ендпоінти для безпечної обробки платежів, і це дозволить оплачувати бронювання прямо через нашу платформу.

ВИСНОВОК

Створений та реалізований веб-додаток, особливо серверна частина, знаходиться на початковій стадії і є що покращувати і куди розвиватися в майбутньому. Як видно з документа, чималий обсяг вже реалізований, але на нас чекає ще багато роботи. Сподіваюся, що при виявленні помилок чи якихось недоліків, згодом їх буде виправлено.

Бекенд-частина фільтрації та бронювання житла була реалізована та протестована. API-ендпоінти, що відповідають за фільтрацію та резервацію, були протестовані за допомогою інструментів Postman, який підтвердив їхню працездатність, та готовність до інтеграції з фронтендом.

Винесений урок — правильне архітектурне планування на стороні сервера. Якщо ми не розробимо контракти API для таких модулів, як фільтрація та резервування, це буде проблематично під час впровадження, оскільки його коригування займе багато часу та ресурсів. Немає потреби критикувати внутрішню архітектуру веб-додатку. Ми використовували .NET Core Framework разом з EF Core ORM, що забезпечило нам незалежність і гнучкість, спростивши оновлення або зміну задіяних технологій.

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 .

6.	MailKit ; Документація: посилання .
7.	AutoMapper; Документація: посилання .
8.	Microsoft SQL; Документація: посилання .
9.	Google Cloud SQL; Документація: посилання .
10.	Microsoft Azure; Документація: посилання .
11.	Google Gemini; Документація: посилання .
12.	GitHub; Документація: посилання .