PROJEKT INŻYNIERSKI

„Aplikacja do ogłoszeń wynajmów krótkoterminowych”

Filip Pyziak
Nr albumu

Kierunek: Informatyka
Specjalność: Bazy Danych i Inżynieria Systemów

PROWADZĄCY PRACĘ
dr hab. inż. Paweł Kasprowski, prof. PŚ
KATEDRA Informatyki Stosowanej
Wydział Automatyki, Elektroniki i Informatyki

GLIWICE 2022/23

Tytuł pracy:
Aplikacja do ogłoszeń wynajmów krótkoterminowych

Streszczenie:
Celem projektu było stworzenie aplikacji internetowej, służącej do dodawania ogłoszeń najmów oraz rezerwowanie tychże ogłoszeń. Głównym założeniem było zastosowanie architektury modularnego monolitu. Ponad to w projekcie wykorzystano podejście *Domain-Driven Design*, a także ideę *Event Sourcingu.*

Słowa kluczowe:
.NET, Aplikacja internetowa, Angular, Domain-Driven Design, Event Sourcing, Modularny Monolit

Thesis title:
Application for short-term rental

Abstract:
The goal of the project was to create a web application, used to add rental ads and book these ads. The main idea was to use modular monolith architecture. In addition, the project used the *Domain-Driven Design* approach, as well as the idea of *Event Sourcing*.

Keywords:
.NET, Web Application, Angular, Domain-Driven Design, Event Sourcing, Modular Monolith

Spis treści

Rozdział 1
Wstęp

W ramach projektu stworzony został portal internetowy, dzięki któremu użytkownicy mają możliwość wynajmu mieszkania na dogodny okres. Jednak to nie wskazana funkcjonalność aplikacji była główną ideą przyświecającą jej powstaniu. Problemem, który miał zostać rozwiązany, było zagadnienie skalowalności oraz łatwego przenoszenia komponentów aplikacji do systemu rozproszonego.

1.1. Cel pracy

Najważniejszym celem budowy tego typu serwisu było zastosowanie odpowiednich narzędzi i architektury – tak, aby aplikacja mogła jak najdłużej radzić sobie z rosnącą liczbą użytkowników. W momencie, gdy przestałby on optymalnie wykonywać przypisane mu funkcje, mógłby zostać przeniesiony na lepiej skalowalny system w sposób wykorzystujący możliwie najmniejszy nakład pracy i ilość zasobów (w tym czasowych czy finansowych). Przykładem takiego systemu jest system rozproszony.

1.2. Zwięzła charakterystyka rozdziałów

- Rozdział 2 – *Analiza tematu* – skupia się na przedstawieniu dostępnych usług istniejących na rynku, służących do najmów krótkoterminowych. W dalszej jego części scharakteryzowane zostały zagadnienia związane z implementacją dobrze skalowalnej architektury.
- Rozdział 3 – *Wymagania i narzędzia* – to zobrazowanie użytkowych i technicznych warunków, jakie musi spełniać stworzony rodzaj systemu. Rozdział zawiera ponadto opis rozwiązań niebędących bezpośrednio związanych z kodem źródłowym aplikacji, w tym Domain-Driven Design, CQRS, event sourcing i modularny monolit.

- Rozdział 4 – *Specyfikacja zewnętrzna* – przedstawia działanie aplikacji od strony klienta. Wprowadza rozróżnienie między kategoriami użytkowników i odnosi się do sfery bezpieczeństwa, której charakterystyka techniczna znajduje się w rozdziale kolejnym. Analizuje również możliwe scenariusze działania, w tym zarządzanie ogłoszeniami, rezerwacjami, recenzjami, a wreszcie samymi kontami. .
- Rozdział 5 – *Specyfikacja wewnętrzna* – zawiera opis poszczególnych modułów aplikacji, w ramach których zaprezentowane zostały warstwy ich architektury. Charakteryzuje wykorzystane w projekcie biblioteki zewnętrzne oraz wewnętrzne. Porusza wskazane wyżej aspekty bezpieczeństwa. Wreszcie, wkupia się na przechowywaniu i wykorzystywaniu danych, w tym ich projekcji oraz synchronizacją między modułami.
- Rozdział 6 – *Weryfikacja i walidacja* – zawiera informacje o testach przeprowadzonych w celu weryfikacji poprawności działania aplikacji. W dalszej części rozważane są napotkane podczas tworzenia projektu błędy oraz zaimplementowane rozwiązania.
- Rozdział 7 – *Podsumowanie i wnioski* – poza informacjami dotyczącymi zrealizowanych zagadnień, uwzględnia dodatkowo napotkane podczas tworzenia oprogramowania problemy oraz charakteryzuje kierunki potencjalnego rozwoju aplikacji.

Rozdział 2
Analiza tematu

2.1. Analiza problemu

Z uwagi na rozpowszechnianie się w ostatnich latach technik cyfrowych i zwiększającą się mobilność społeczeństwa, w istotnym stopniu wzrosła popularność stron internetowych i aplikacji mobilnych służących do najmu nieruchomości, w tym w szczególności najmu krótkookresowego. Pierwsza tego rodzaju strona powstała w 1996 roku – Bookings.nl, holenderski serwis założony przez Geert–Jan Bruinsma. Początkowo wykorzystywany był on jedynie w sferze usług hotelarskich [1]. Aktualnie witryna ta jest szerzej rozpoznawalna pod adresem nowej domeny – Booking.com. Niemniej prawdziwą rewolucję zapewniło dopiero stworzenie usługi internetowej Airbnb w roku 2008. W tym przypadku podstawowym zamysłem było opracowanie platformy, która umożliwiałaby dodawanie ogłoszeń najmu krótkoterminowego osobom prywatnym. Zrewolucjonizowało to rynek turystyczny i nieruchomości. Wskazane serwisy dysponują łącznie ponad 60% udziałem w rynku, w tym na Booking.com przypada 34,57% udziału, a na Airbnb – 25,97% [2]. Dla unaocznienia skali zasięgu – Booking.com szczyci się liczbą aktywnych ogłoszeń w wysokości 6,6 milionów, a Airbnb oferuje ich 6 milionów [3].

Przełożenie na sferę najmów mają również ogólnoświatowe kryzysy gospodarcze, takie jak te będące konsekwencją pandemii SARS-CoV-2 czy inwazji Rosji na Ukrainę. Nie bez znaczenia pozostają kryzysy lokalne determinujące zamożność portfeli mieszkańców danego państwa. Wyraźnie widać to na rynku polskim. W ostatnich latach zarysowuje się tendencja odchodzenia od kupna mieszkań z uwzględnieniem perspektywy długoterminowej na rzecz najmów krótko- lub długookresowych. Dodatkową niepewność niesie przyszła sytuacja społeczna, polityczna czy gospodarcza w regionie. Popularność takich stron jak Morizon.pl istotnie wzrosła w ostatnich latach. Do najmu wykorzystuje się aktualnie również serwisy domyślnie nieprzystosowane do tego, takie jak Olx.pl, Facebook.com.

Rosnąca popularność takich usług przekłada się na zwiększone wykorzystanie mocy obliczeniowej serwerów oraz przepustowości sieci. To również bezpośrednio wiąże się ze spadkiem wydajności systemów i wzrostem kosztów ich utrzymania. Wynika to z faktu, że zazwyczaj systemy te są oparte na architekturze klasycznego monolitu, który dobrze skaluje się wertykalnie, ale ma niską skalowalność horyzontalną. Skalowanie wertykalne oznacza, że aby ulepszyć sprawność działania całego systemu należy dołoży lub wymienić podzespoły takie jak RAM, dyski lub procesory. Z kolei skalowanie horyzontalne ma na celu dodanie kolejnych, niezależnych od siebie maszyn. W klasycznym monolicie, należy za każdym razem uruchomić cały system na nowej jednostce, co znaczy, że nie może on być skalowany tylko tam, gdzie jest najbardziej obciążony.

2.2. Propozycja rozwiązania problemu

Rozwiązaniem tego typu problemów może być zastosowanie architektury mikroserwisów, która w ostatnim czasie zdominowała świat IT [4]. Jednak jej poprawna implementacja wymaga odpowiedniego zaplecza – doświadczenia, właściwej wiedzy o systemach rozproszonych, a także sporego kapitału, na który nie każda korporacja jest w stanie sobie pozwolić. Migracja z klasycznego monolitu do systemu opartego na mikrousługach wydaje się bardzo trudna a w przypadku niektórych projektów wręcz niemożliwa. Zdarza się, że mimo wystarczającego finansowania oraz zatrudnienia szeregu ekspertów przejście na mikroserwisy niczego nie rozwiązuje. Staje się to tylko kolejnym problemem w Wielkiej Kuli Błota (z ang. Big Ball of Mud) [5]. Wbrew powszechnym poglądom architektura ta w większości przypadków nie dostarcza wymiernych korzyści, a wręcz przeciwnie – problemy związane z systemami rozproszonymi całkowicie niwelują jakiekolwiek benefity z posiadania mikrousług.

Alternatywnym rozwiązaniem jest tworzenie oprogramowania w oparciu o architekturę modularnego monolitu. Według wielu takie rozwiązanie łączy ze sobą najlepsze cechy klasycznego monolitu oraz skalowalnych mikroserwisów, co powoduje, że to rozwiązanie świetnie sprawdza się dla małych i średnich systemów, a w wypadku rozrośnięcia się oprogramowania do wielkich rozmiarów umożliwia płynne przejście na architekturę mikroserwisów w sposób łatwy dla programistów oraz przy całkiem niedużych kosztach wdrożenia [6].

Jednym z ważniejszych aspektów systemów rozproszonych jest to, by kolejne jego komponenty nie były ze sobą silnie sprzężone. Tradycyjne podejście do tworzenia aplikacji, czyli *Model-Driven Engineering* tego niestety nie ułatwia. Z pomocą przychodzi metodologia *Domain-Driven Design (DDD),* która wyodrębniania poszczególne problemy biznesowe, a także umieszcza je w odpowiednich modułach. Oprócz tego problem

skalowalności może zostać rozwiązany wykorzystując wzorzec *CQRS*, który rozdziela polecenia od zapytań. W celu utrzymania ciągłości systemu, a także zachowania danych historycznych, zastosowane może zostać rozwiązanie *Event Sourcing* – mechanizm pozyskiwania zdarzeń. Zagadnienia te zostały dogłębniej opisane w następnym rozdziale.

Rozdział 3
Wymagania i narzędzia

3.1. Wymagania funkcjonalne

Podstawowe wymagania funkcjonalne aplikacji służącej do wynajmów krótkoterminowych:

- Tworzenie nowych kont dla użytkowników ogłaszających wynajem miejsca
- Tworzenie nowych kont dla użytkowników chcących rezerwować miejsca na określony termin
- Tworzenie kont moderatorów
- Blokowanie kont użytkowników przez uprawnionych do tego moderatorów
- Przeglądarka użytkowników
- Logowanie użytkowników przy pomocy e–maila oraz hasła
- Edycja danych konta użytkownika
- Dodawanie ogłoszeń o wynajmie tylko przez konta do tego przeznaczone, wraz z tytułem, opisem, adresem (ulica, miasto, kraj), zdjęciami, informacjami kontaktowymi właściciela oraz ceną
- Wyszukiwarka ogłoszeń, wraz z paginacją oraz możliwością filtrowania ogłoszeń w zależności od kraju, miasta lub dat w których dane miejsce posiada jeszcze wolne terminy
- Edycja ogłoszenia o wynajmie, dostępna tylko dla autorów ogłoszenia. Pozwalająca na zmianę tytułu, opisu, ceny za noc, dodanie nowych zdjęć, usunięcie wybranych zdjęć, adresu (ulicy, miasta, kraju)
- Przeglądarka dodanych przez użytkownika ogłoszeń
- Usuwanie dodanych ogłoszeń
- Podgląd detaliczny danego ogłoszenia wraz z dodanymi recenzjami na temat miejsca

- Rezerwacja miejsca przez konto do tego wyznaczone, w wybranym przez siebie terminie. Brak możliwości rezerwacji terminów zajętych przez innych użytkowników oraz tych, które już upłynęły
- Przeglądarka dokonanych rezerwacji.
- Możliwość dodania recenzji dla rezerwacji, których termin trwania już minął
- Anulowanie rezerwacji

3.2. Wymagania niefunkcjonalne

Podstawowe wymagania niefunkcjonalne aplikacji służącej do wynajmów krótkoterminowych:

- Dostęp do aplikacji 24 godziny na dobę, z wyłączeniem przerw na prace związane z utrzymaniem aplikacji
- Brak ograniczeń związanych z lokalizacją użytkownika
- Poprawna kompatybilność aplikacji wraz ze wszystkimi nowoczesnymi przeglądarkami
- Łatwość zmiany środowiska aplikacji dzięki wykorzystaniu konteneryzacji
- Brak błędów uniemożliwiających korzystanie z aplikacji
- Łatwy w użytkowaniu interfejs graficzny
- Architektura aplikacji w oparciu o modularny monolit
- Wykorzystanie podejścia *Domain–Driven Design* oraz *Event Sourcing*
- Spójna kolorystyka

3.3. Domain–Driven Design

Wraz z kolejnymi iteracjami naszego oprogramowania coraz trudniej jest utrzymać spójność kodu tak, by nie dochodziło do niepotrzebnego przeniesienia odpowiedzialności i logiki do nieodpowiednich komponentów. Z pomocą przychodzi opisana przez Erica Evansa metodologia *Domain–Driven Design* [7]. Skupia się ona głownie wokół zagadnień biznesowych i na oddzieleniu ich od aspektów technicznych aplikacji. Jej głównym założeniem jest stworzenie sfery domeny, która oznacza zbiór pewnych aktywności i wiedzy potrzebnych do poprawnego określenia wymagań, terminologii i funkcjonalności

dla logiki biznesowej. W rezultacie programista jest w stanie ograniczyć złożoność oprogramowania. Podejście to wymaga także bezpośredniej współpracy programistów z klientami i używania przez obie strony wszechobecnego języka (z ang. *Ubiquitous Language*).

Oprócz rozdzielenia warstwy biznesowej od warstw infrastruktury, wydzielane powinny być także konteksty ograniczone (z ang. *Bounded Context*). Polega to na podzieleniu logiki biznesowej w taki sposób, by konteksty były od siebie niezależne, a język wykorzystywany w nich powinien być jednoznaczny. Najlepiej zobrazować to zagadnienie na przykładzie. W projekcie została wydzielona warstwa odpowiedzialna za dodawanie ogłoszeń wynajmu oraz warstwa służąca do uwierzytelnianie kont użytkowników. Gdyby warstwy nie zostały rozdzielone istniałby jeden wspólny model użytkownika przypisywany do każdego ogłoszenia oraz służący do uwierzytelnienia. Niestety w tym wypadku nie ma możliwości odpowiednio precyzyjnie nazwać tego modelu tak, by był on reprezentowany przez oba konteksty. Jednak w przypadku, gdy konteksty są od siebie rozdzielone, to w jednym z nich powstanie model właściciela ogłoszenia, a w drugim model konta użytkownika. Taka precyzyjność pozwala na łatwiejsze zrozumienie kodu oraz ułatwia dodawanie niezależnych od siebie funkcjonalności. Wydzielenie kontekstów nie oznacza, że muszą one być od siebie w pełni niezależne, jednak pozwala zaobserwować istnienie tych powiązań w bardziej przejrzysty sposób.

Model domenowy składa się z obiektów wartościowych (z ang. *Value Objects*), które odzwierciedlają właściwości obiektu. Istnieją one tylko w obrębie jednego agregatu (pojęcie to zostało opisane w późniejszym akapicie) i są niezmienne (z ang. *Immutable*) co oznacza, że zamiast uaktualniać istniejące obiekty są one zastępowane nowymi instancjami. Utworzenie obiektu wartościowego zależy od reguł biznesowych. Oznacza to, że jeśli w aplikacji opis ogłoszenia wynajmu nie może przekraczać 500 znaków, to stworzenie takiego obiektu będzie zablokowane przez odpowiednią walidację umieszczoną w logice tworzenia tego obiektu.

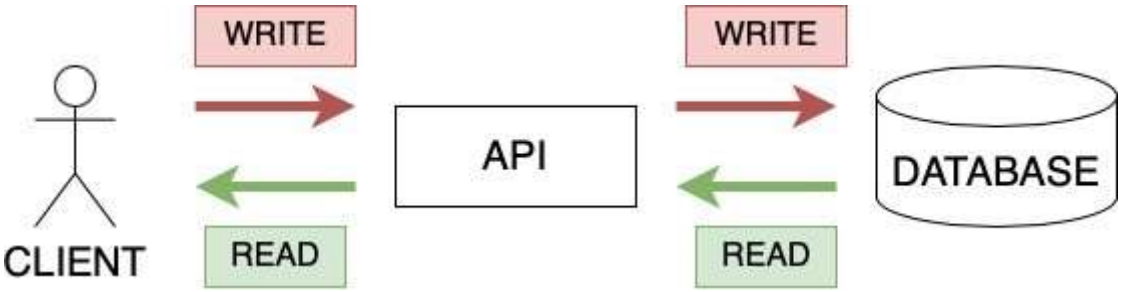
Kolejną częścią obiektu domenowego jest encja (z ang. *Entity*) czyli zbiór obiektów wartościowych tworzących spójnie logiczną całość oraz funkcje definiujące jego stan. Mogą one zmieniać stan swoich właściwości pozostając unikatowe w obrębie agregatu (z ang. *Aggregate*), który jest następnym elementem modelu domenowego. Jest on zbiorem encji oraz obiektów wartościowych, wyznacza on granice między encjami. Posiada on także specjalną encje nazywaną korzeniem agregatu (z ang. *AggregateRoot*), do której jako jedynej odwoływać mogą się inne obiekty z zewnątrz [8, 9, 10]. W niektórych przypadkach w modelu biznesowym istnieje potrzeba wywołania pewnej logiki, nie do końca związanej ze szczególnym obiektem wartościowym lub encja. W takim przypadku logika jest umieszczana w klasie nazywanej serwisem domenowym (z ang. *Domain Service*) [11].

3.4. CQRS

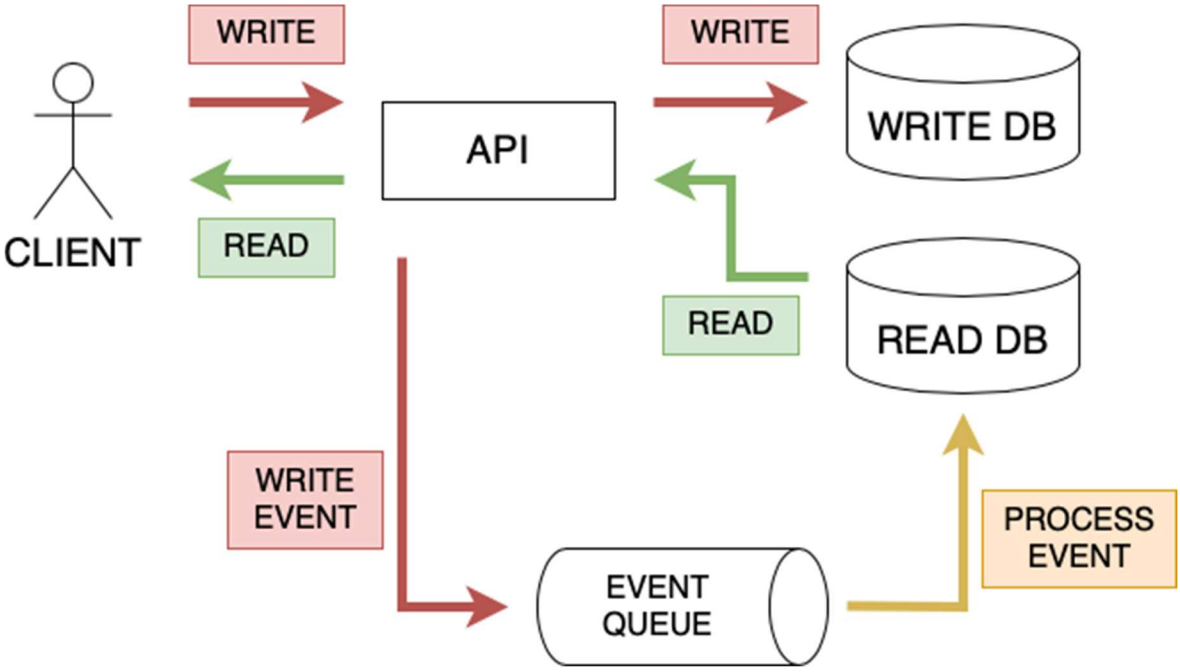
W wypadku tradycyjnego podejścia aplikacja wykorzystuje tę samą bazę danych do wykonywania poleceń (operacji zapisu) i zapytań (operacji odczytu) - patrz Rys.1. Tworzy to olbrzymi problem, gdyż wraz ze wzrostem popularności aplikacji można zaobserwować dysproporcję w ilości zapytań i poleceń. W większości to zapytania zdominują aplikację, dlatego potrzebne będzie rozwiązanie przyspieszające odczyt danych, jednak oczywiście zdarzają się odwrotne sytuacje np. w systemach *Internet of Things.*

Wykorzystanie wzorca *CQRS – Command and Query Responsibility Segregation* autorstwa Greg Younga [12] polega na oddzieleniu zapytań od poleceń. W ten sposób, aplikacja może np. zrezygnować z korzystania ze wspólnej bazy danych. Zamiast tego może ona zostać rozdzielona na osobne bazy dla operacji zapisu i operacji odczytu (patrz Rys.2.). Pozwala to na wykorzystaniu optymalnych rozwiązań dla poszczególnych operacji.

W pokazanym na rysunku nr. 2 przykładzie przy wykonywaniu poleceń dane są zapisywane bezpośrednio w bazie do tego przeznaczonej w sposób synchroniczny, a do specjalnego strumienia zdarzeń dodawane jest odpowiednie zdarzenie odpowiadające danym uprzednio zapisanym. Następnie zdarzenie to jest odpowiednio procesowane w sposób asynchroniczny i gdy nadejdzie jego kolej, dane te są zapisywane w bazie przeznaczonej do odczytu. W taki sposób można uzyskać rozwiązanie bardzo dobrze skalowalne z wysoką dostępnością danych. Z uwagi na asynchroniczny wymiar tego rozwiązania nie jesteśmy w stanie zapewnić spójności odczytywanych i zapisywanych danych. Jednym z rozwiązań tego problemu jest odczytywanie danych z bazy poleceń przez autorów tych poleceń. Wzorzec świetnie sprawdza się dla rozwiązania *Event Sourcing* [13, 14].



Rysunek 1. Klasyczny model zapisu danych



Rysunek 2. Przykładowy model zapisu danych z wykorzystaniem wzorca CQRS

3.5. Event sourcing

Pojęcie *Event Sourcing* zostało wprowadzone przez Grega Younga [15]. Polega na tym, że stan encji naszej aplikacji nie jest utrwalany, lecz rekonstruowany z wydarzeń, które miały miejsce do tej pory. Rekonstrukcja wykonywana jest chronologicznie. Stan encji reprezentuje dane w danej chwili. Analogiczne działanie możemy zaobserwować w systemach bankowych, gdzie saldo konta jest budowane na podstawie wszystkich dotychczas dokonanych transakcji. Wydarzenia odzwierciedlają wszystkie zmiany stanu danych dla danej encji.

Pozornie może wydawać się to skomplikowane, dlatego najlepiej jest to zobrazować na przykładzie. W bazie danych umieszczony jest użytkownik zawierający pole login i hasło. W sytuacji, gdy nie wykorzystywany jest *Event Sourcing*, a hasło zostanie zmienione, dane w bazie zostaną zaktualizowane, a poprzedni stan na zawsze zostanie utracony. W przypadku wykorzystania *Event Sourcing* zamiast informacji o konkretnym stanie będziemy przechowywać dane o zmianie tego stanu. Oznacza to, że w momencie zmiany hasła w historii dalej będą widniały zdarzenia poprzedzające (np.: utworzenie użytkownika lub uprzednia zmiana danych). Jako baza danych może posłużyć dowolne bazodanowe rozwiązanie dostępne na rynku. Najczęściej jednak korzysta się z dedykowanych w celu utrwalania zdarzeń strumieniowych baz danych (np. *Event Store DB, Apache Kafka,*

Amazon Kinesis Streams), które przechowuje wszystkie dotychczasowe zdarzenia w tzw. strumieniach. Strumienie te posiadają swoje unikatowe identyfikatory. Mogą one reprezentować wspominane w podrozdziale *Domain–Driven Design* agregaty. Obiekt w danej chwili jest rekonstruowany ze wszystkich zdarzeń i jego właściwości będą dokładnie takie same jak w przypadku tradycyjnego podejścia. Jednak z dostępem do pełnej historii zmiany stanów.

Można zaobserwować, że w tym wypadku idealnie sprawdza się wzorzec *CQRS*. Polecenia zmieniające stan encji wywołują zdarzenia zmiany stanu, które następnie zapisywane są w magazynie zdarzeń, oraz tworzona jest swego rodzaju migawka aktualnego stanu tych encji w osobnej bazie danych (np.: tradycyjna relacyjna baza Sql, baza dokumentowa, cache), za pomocą specjalnych projekcji wywoływanych wraz ze zdarzeniem. Problemem może wydać się odczytywanie zdarzeń ze wszystkich strumieni za każdym razem, gdy pobierany jest dany korzeń agregatu. Na szczęście w mechanizmie pozyskiwania zdarzeń istnieje na to pewne rozwiązanie, nazywane punktem kontrolnym (z ang. *Checkpoint*), który pozwala na to, aby po restarcie aplikacji zdarzenia nie odtwarzały się od początku istnienia. Gdy punkt kontrolny zostanie usunięty, wszystkie dotychczasowe zdarzenia zostaną chronologicznie wywołane na nowo, a on sam zostanie utworzony na nowo. Pozwala to na przykład na stosunkowo łatwą (w przypadku dużej aplikacji odtwarzanie projekcji może zająć kilka dni, a nawet tygodni) zmianę silnika bazy danych służącej do odczytu [16]. Innym problem jest read–after–write, pojawia się on, gdy baza danych odczytu jest bardzo obciążona. W konsekwencji czego mamy inny stan obiektu w bazie zapisu i inny w bazie odczytu. Dochodzi wtedy do sytuacji, w której użytkownik nie widzi swoich zmian w interfejsie, mimo że zostały przez niego wykonane. Jednym z rozwiązań takiego problemu synchronizacji jest odczytywanie danych przez wykonawcę wspomnianej wcześniej operacji bezpośrednio z bazy służącej do zapisu [17].

3.6. Modularny monolit

W odróżnienia od klasycznego systemu monolitycznego modularny monolit składa się jak sama jego nazwa mówi z modułów. Każdy moduł powinien być przez nas traktowany jako osobna aplikacja. Sprawia to, że są one autonomiczne i luźno lub nawet wcale nie sprzężone między sobą. W praktyce umożliwia to pracę nad rozwojem każdego z modułów osobnemu zespołowi, podobnie jak w przypadku mikroserwisów. Każdy z modułów powinien być skupiony na osobnej poddziedzinie aplikacji tworząc wspólnie jej domenę.

Kolejną różnicą jest wykorzystanie baz danych przez naszą aplikację. Celem wzajemnego uniezależnienia każdego z modułów nie należy korzystać z wspólnej bazy

danych. Powinna ona być unikalna dla każdego modułu, a gdy dane są współdzielone powinny one być duplikowane oraz synchronizowane asynchronicznie przez zdarzenia integracyjne lub synchronicznie tak, aby jeden moduł odpytywał drugi o szczegółowe dane. Takie podejście do monolitu zapewni nam łatwość rozwijania naszej aplikacji i ewentualną prostą migrację do systemu rozproszonego [18, 19].

Rozdział 4
Specyfikacja zewnętrzna

4.1. Obsługa i uruchomienie

Do uruchomienia w środowisku produkcyjnym niezbędna jest oprogramowanie Docker służące do wirtualizacji kontenerów. By poprawnie uruchomić aplikację, należy znaleźć się w głównym folderze aplikacji, gdzie znajduje się plik *docker–compose.yml.* Kolejnym krokiem jest uruchomienie w terminalu komendy *docker compose up.* Po wykonaniu tego kroku wszystkie kontenery powinny uruchomić się automatycznie, może to jednak chwilę zająć. Aplikacja od tego momentu powinna działać poprawnie.

Część backend-owa projektu działa pod adresem: *http://localhost:5000*. Natomiast frontend pod adresem *http://localhost:4200*. Aplikacja była uruchamiana i testowana na systemie operacyjnym Windows 10 wraz z Dockerem w wersji v4.13.0, aczkolwiek nie wyklucza to wykorzystania innej wersji oprogramowania.

4.2. Kategorie użytkowników

W aplikacji istnieją trzy typy kont. Konto moderatora odpowiada za zarządzanie innymi kontami. Jest ono tworzone wraz ze startem oprogramowania, a dane do logowania prezentują się następująco login – *admin@admin.pl* i hasło – *admin*. Kolejnym typem jest konto właściciela miejsca do najmu – *Place owner*. Użytkownicy ci są w stanie dodawać nowe ogłoszenia najmu i zarządzać nimi. Ostatnim rodzajem jest typ najemcy – *Tenant,* do jego funkcjonalności należy składanie rezerwacji oraz posiada możliwość recenzowania tych, które są już zakończone. Jedno konto może posiadać tylko jeden typ i nie ma możliwości jego zmiany.

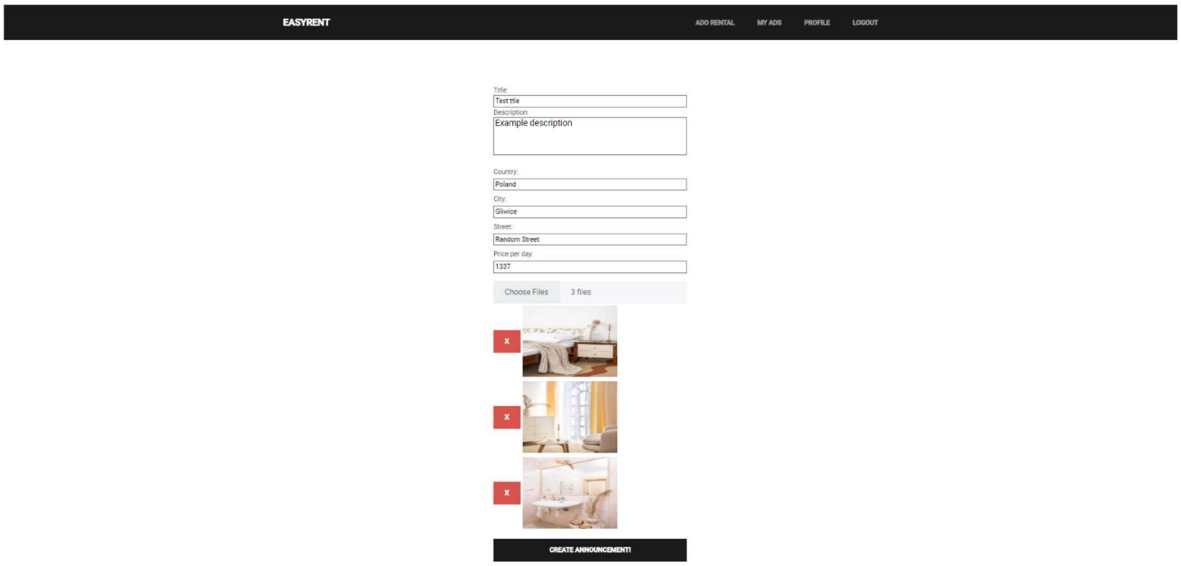
4.3. Bezpieczeństwo

W aplikacji wykorzystano *Cross–origin resource sharing (CORS)* w taki sposób, by dostęp do zasobów serwera był ograniczony tylko dla domeny części klienckiej. Do autoryzacji użytkowników wykorzystywany jest JWT. Natomiast magazynowanie haseł zabezpieczone zostało wykorzystując haszowanie. Oba te zagadnienia zostały opisane w rozdziale 5.

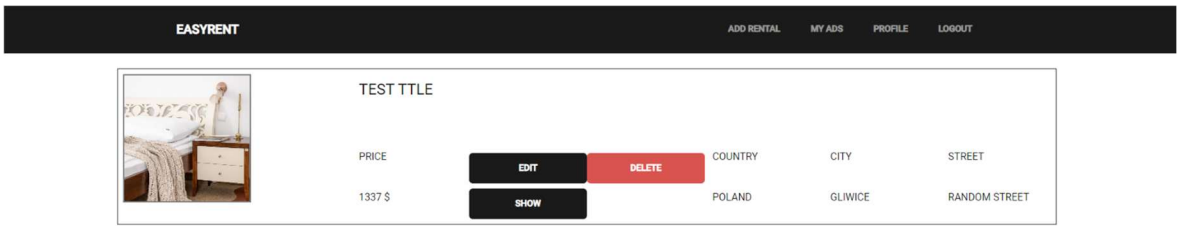
4.4. Scenariusze działania aplikacji

4.4.1. Tworzenie ogłoszenia

Aby utworzyć nowe ogłoszenie należy najpierw zalogować się na konto typu *place owner*. Następnie z paska nawigacji wybrać zakładkę *add rental.* Po naciśnięciu odpowiedniego przycisku następuje przekierowanie do formularza tworzenia ogłoszenia najmu (patrz Rys. 3.). Należy go wypełnić odpowiednimi danymi zgodnie z opisem poszczególnych rubryk. Możliwe jest także dodanie zdjęć w formacie JPG lub PNG. Po wciśnięciu przycisku *create announcement,* jeśli dane były poprawne ogłoszenie zostanie utworzone oraz zostaniemy przeniesieni do zakładki „my ads” (patrz Rys. 4.). W przeciwnym wypadku zostanie wyświetlony komunikat o błędzie.



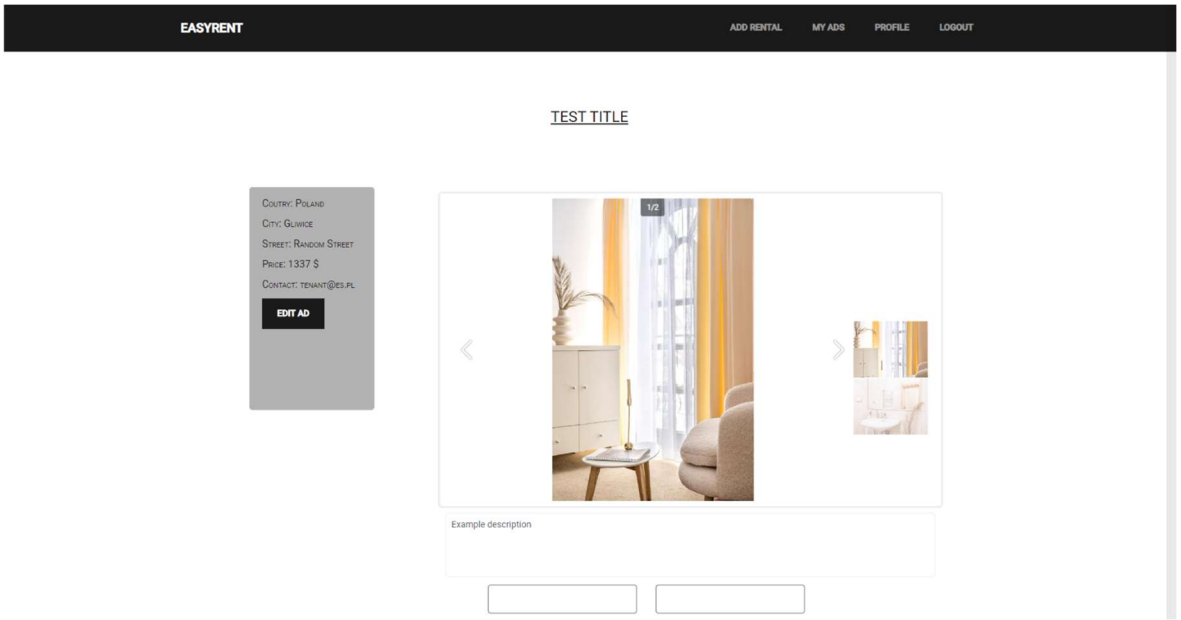
Rysunek 3. Formularz tworzenia ogłoszenia najmu



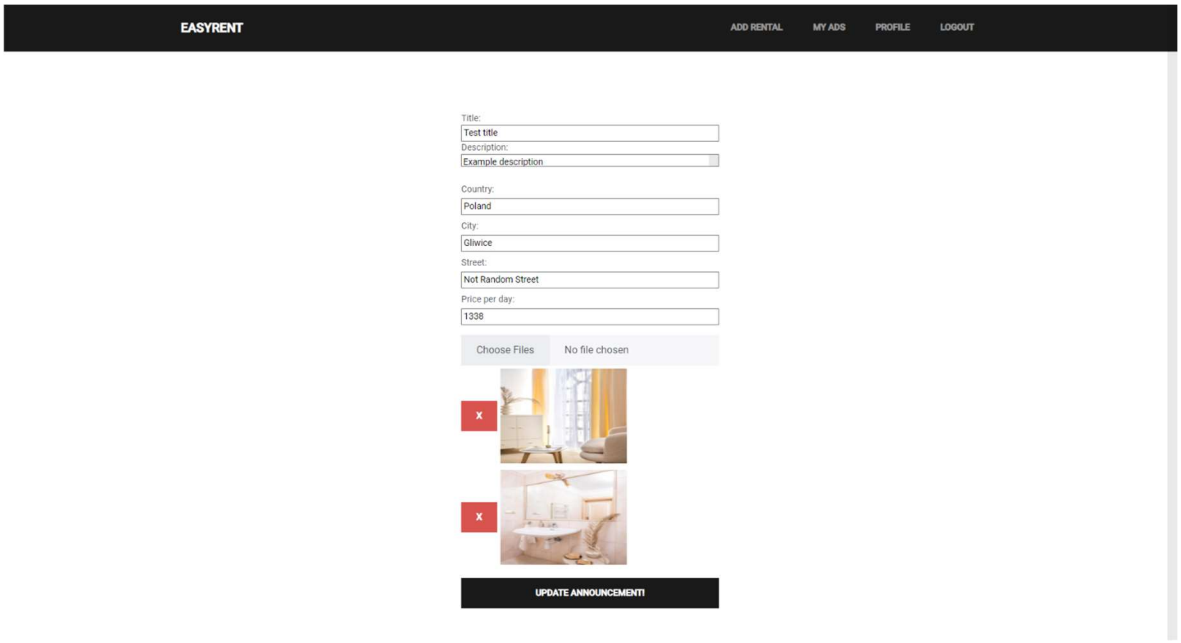
Rysunek 4. Panel moje ogłoszenia

4.4.2. Edycja ogłoszenia

Aby edytować ogłoszenie, jego właściciel musi nacisnąć przycisk *Edit* znajdujący się na wybranym ogłoszeniu w zakładce *my ads* (patrz Rys. 4.) lub bezpośrednio w szczegółach ogłoszenia (patrz Rys. 5.). Po wykonaniu tej czynności użytkownik zostanie przekierowany do widoku edycji (patrz Rys. 6.). Znajdujący się tutaj formularz działa analogicznie do tego opisanego w scenariuszu tworzenia nowego ogłoszenia. Po zatwierdzeniu zmian przyciskiem *Update announcement* użytkownik otrzyma odpowiedni komunikat w prawym dolnym rogu ekranu o sukcesie danej zmiany lub o błędzie, jeśli taki miał miejsce.



Rysunek 5. Szczegółowy widok ogłoszenia



Rysunek 6. Formularz edycji ogłoszenia

4.4.3. Przeglądanie ogłoszeń

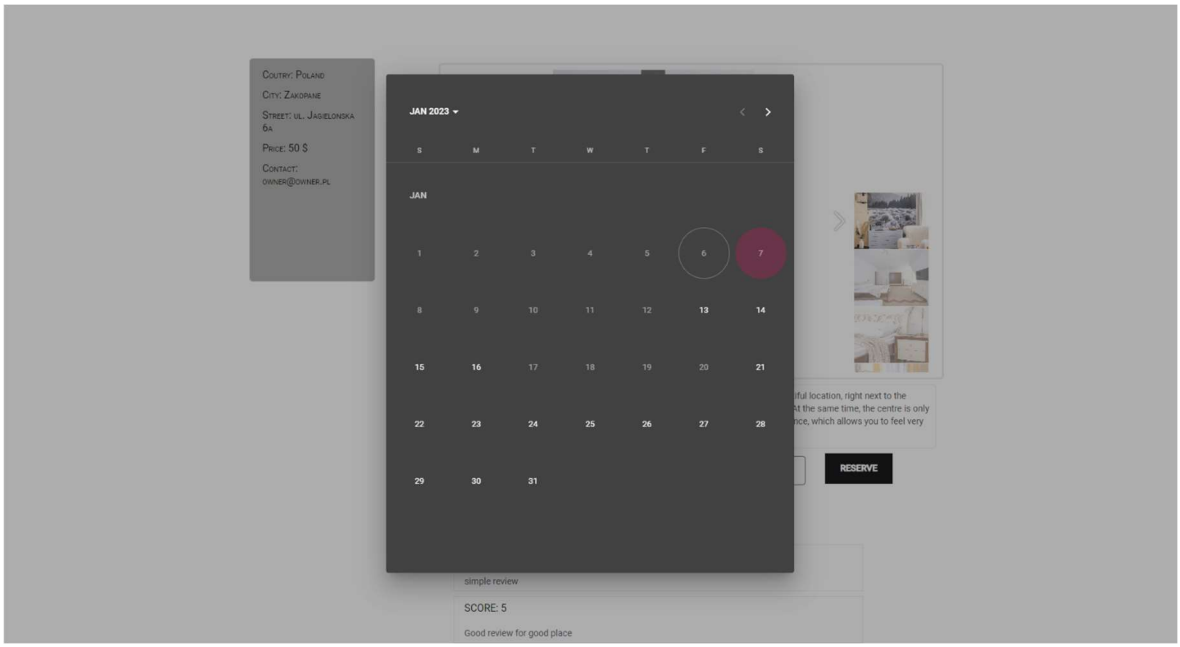
Dostęp do wyszukiwarki posiada każdy użytkownik aplikacji, nawet taki, który nie jest zalogowany. By się do niej dostać należy nacisnąć logo aplikacji znajdującą się po lewej stronie paska nawigacji. Filtrowanie jest możliwe po kraju, mieście (patrz Rys. 7.), a także po wolnych terminach (patrz Rys. 8.). Aby przejść do szczegółów danego ogłoszenia wystarczy w nie kliknąć. Wyszukiwarka posiada paginację, która umożliwia przejście na następną, poprzednią, ostatnią oraz pierwszą stronę.

Rysunek 7. Wyszukiwanie ogłoszeń z filtrem miast

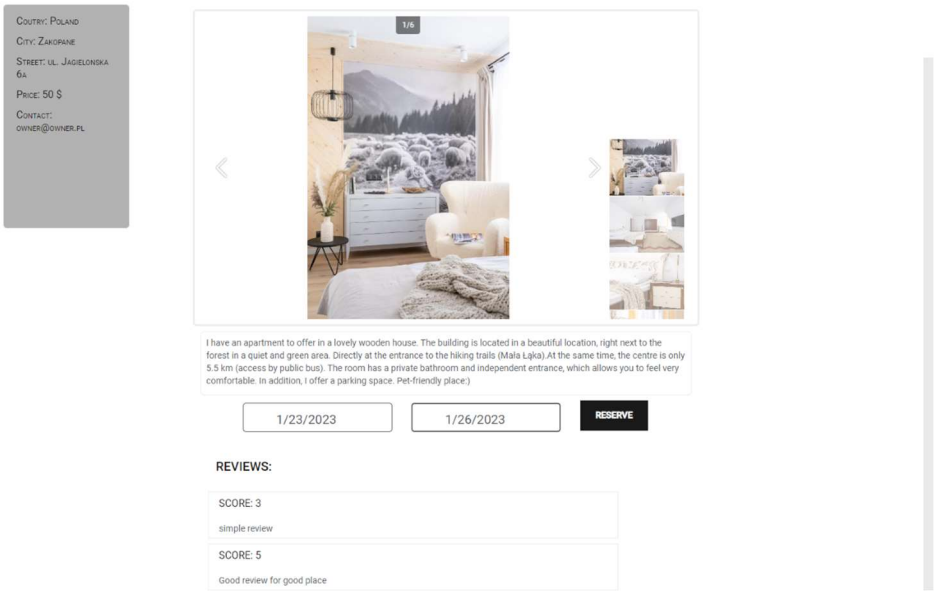
Rysunek 8. Wyszukiwarka ogłoszeń z wykorzystaniem wolnych terminów

4.4.4. Składanie rezerwacji

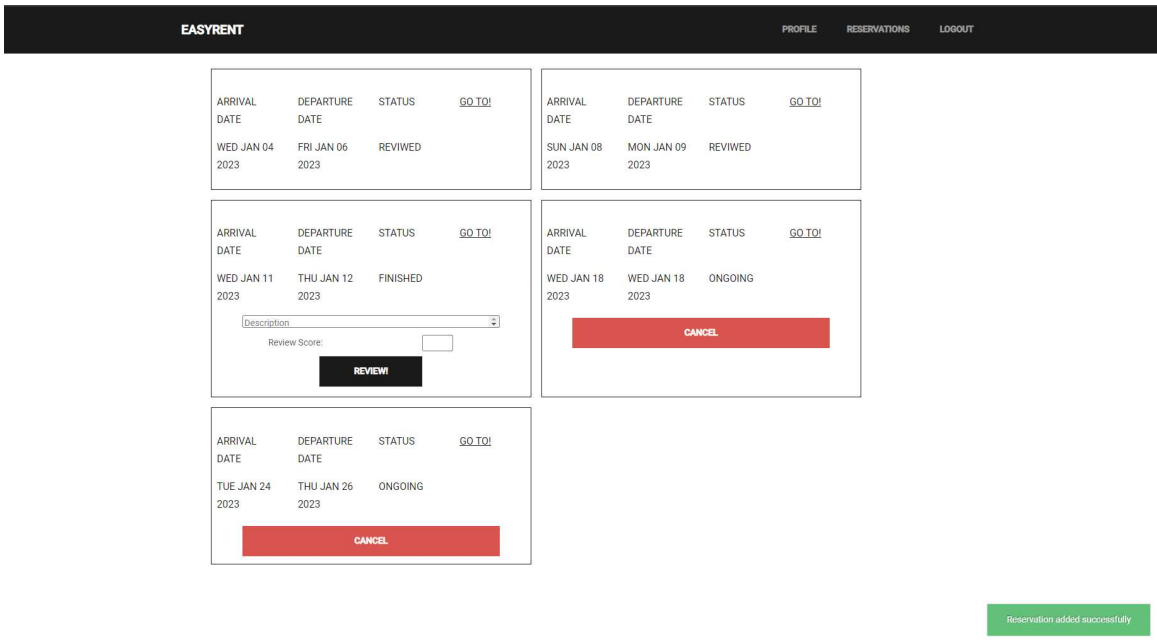
By dokonać rezerwacji należy być zalogowanym na koncie typu *tenant.* Następnie trzeba dokonać wyboru miejsca oraz przenieść się do jego szczegółowego widoku klikając na ogłoszenie. Teraz wystarczy nacisnąć odpowiednią rubrykę *arrival date oraz departure date*. Po ich naciśnięciu wyświetli się okno z wolnymi datami (patrz Rys. 9.). Po wybraniu odpowiednich terminów należy przycisnąć przycisk *reserve* (patrz Rys. 10). Jeśli podane daty były poprawne użytkownik zostanie przekierowany do panelu swoich rezerwacji (patrz Rys. 11.) wraz z odpowiednim powiadomieniem. W przeciwnym wypadku zostanie wyświetlone powiadomienie o błędzie w prawym dolnym rogu ekranu.



Rysunek 9. Panel do wyboru wolnego terminu



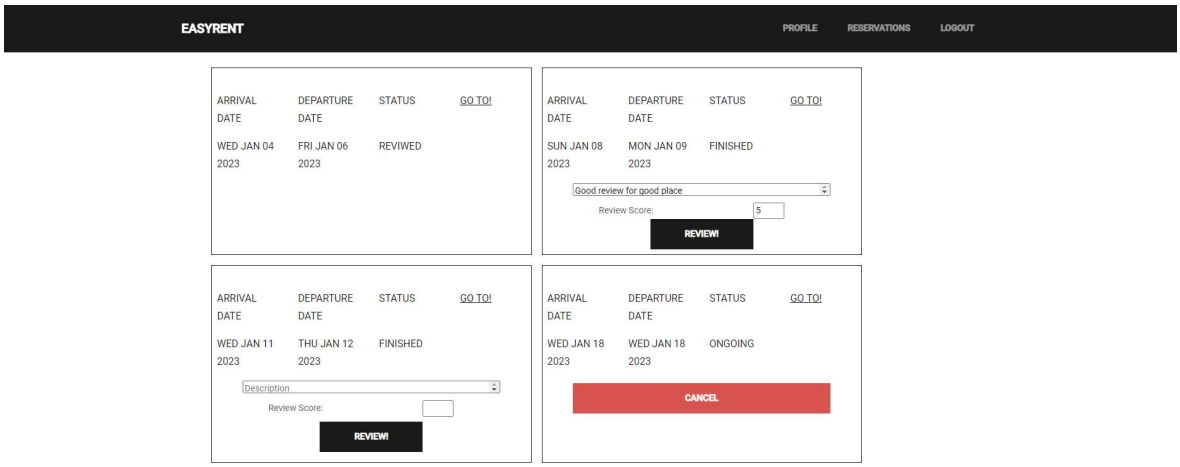
Rysunek 10. Widok detaliczny strony wraz z wybranym terminem rezerwacji



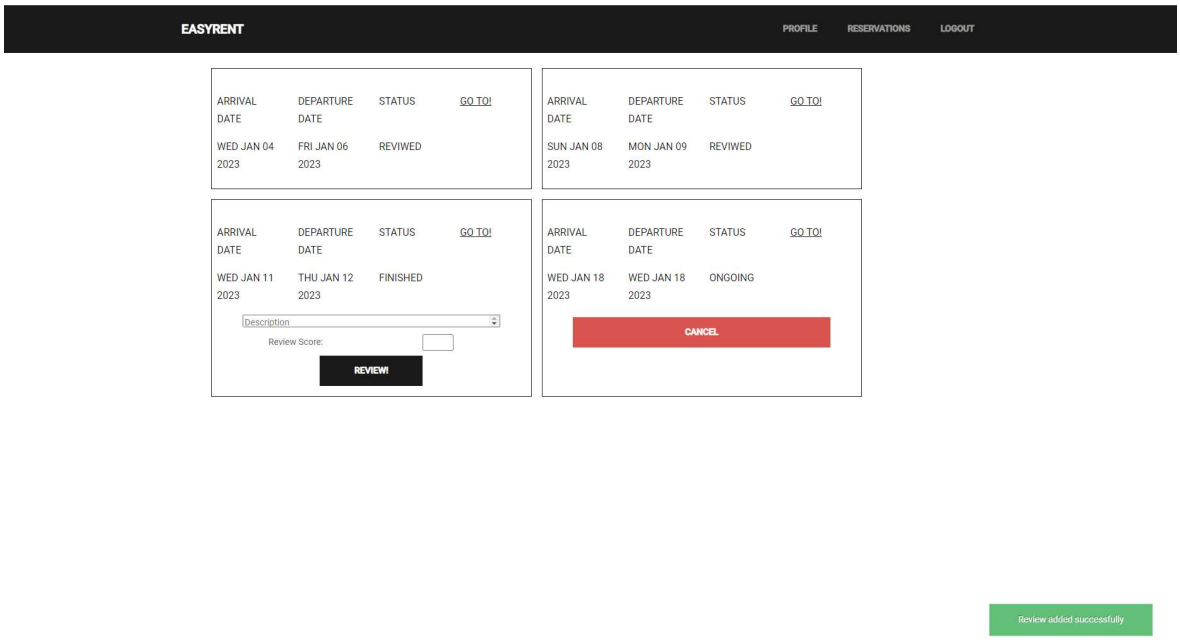
Rysunek 11. Panel moich rezerwacji wraz z komunikatem o udanym komunikatem o dodaniu nowej rezerwacji

4.4.5. Dodawanie recenzji

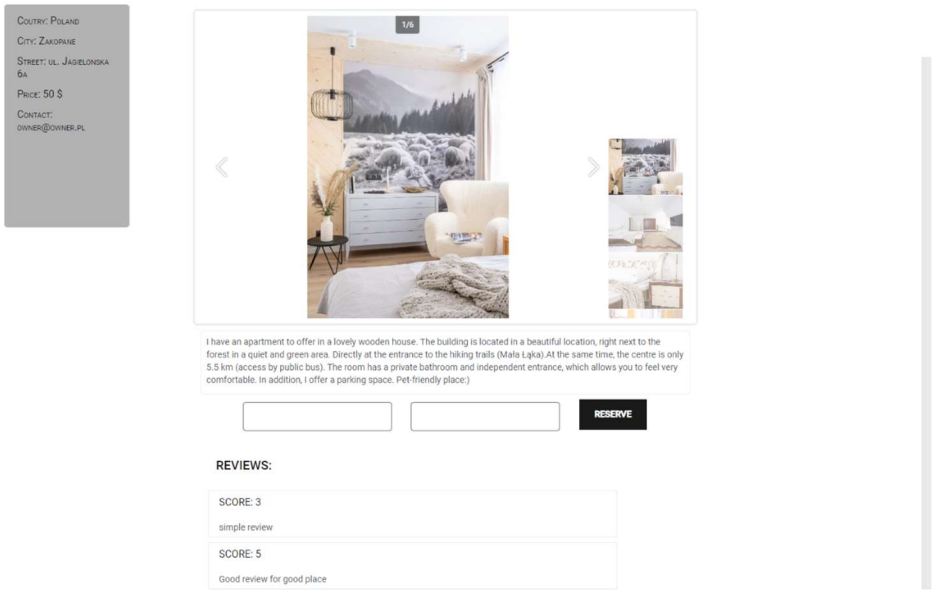
Gdy rezerwacja zostanie zakończona użytkownik otrzymuje funkcjonalność dodania recenzji wraz z oceną punktową swojego pobytu. By to zrobić musi on udać się do panelu rezerwacji za pomocą przycisku *reservations* znajdującego się na pasku nawigacji. Po wyświetleniu panelu (patrz Rys. 12.), na ukończonych rezerwacjach będzie widniał status *finished* oraz pojawią się dwa pola – pierwsze na wpisanie opisu recenzji i drugie na wpisanie oceny w skali od 1–5. Wraz z formularzem pojawia się również przycisk *review,* którego naciśnięcie spowoduje wyświetlenie odpowiedniego powiadomienia (patrz Rys. 13.) oraz dodanie recenzji. W konsekwencji czego z wybranej rezerwacji w panelu zniknie formularz, a status zmieni się na *reviewed.* W widoku szczegółowym ogłoszenia będzie się teraz znajdować dodana recenzja (patrz Rys. 14.).



Rysunek 12. Panel moich rezerwacji z wypełnionym formularzem recenzji



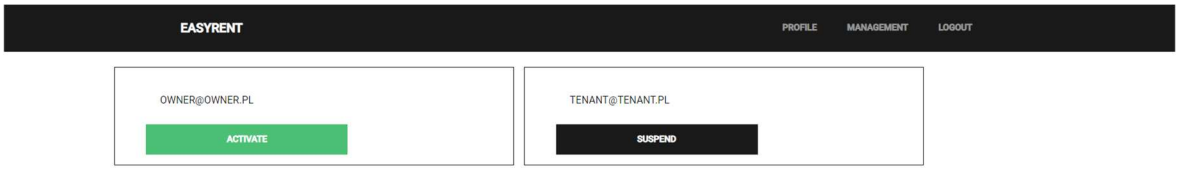
Rysunek 13. Panel moich rezerwacji, po poprawnym dodaniu nowej recenzji



Rysunek 14. Szczegółowy widok ogłoszenia wraz z nowo dodaną recenzją

4.4.6. Blokowanie użytkowników

Aby zablokować konto dowolnego użytkownika w systemie w pierwszej kolejności należy zalogować się na konto moderatora. Po zalogowaniu z paska nawigacji należy wybrać przycisk *management* i przenieść się do panelu zarządzania (patrz Rys. 15.). Teraz należy wybrać użytkownika, którego konto ma zostać zawieszone i wcisnąć przycisk *suspend*. Po tej czynności wybrany użytkownik nie będzie posiadał możliwości zalogowania się do systemu. By go odblokować należy nacisnąć przycisk *active.* Warto tutaj dodać, że konto administratora nie jest w stanie zablokować samego siebie.



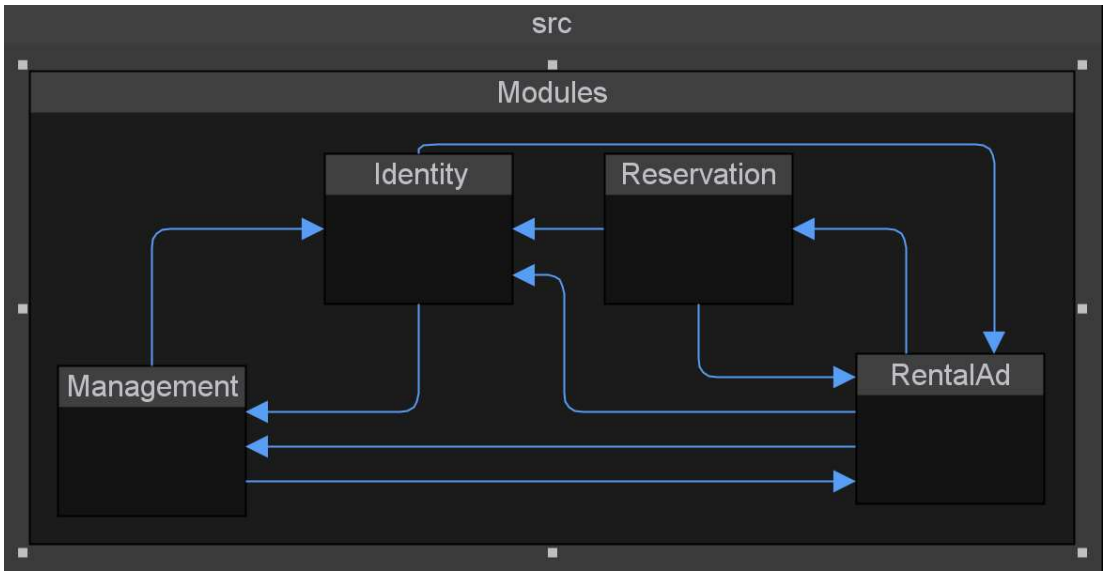
Rysunek 15. Panel zarządzania użytkownikami

Rozdział 5
Specyfikacja wewnętrzna

5.1. Opis modułów

Na etapie tworzenia zbioru funkcjonalności wydzielone zostały cztery osobne konteksty. Część danych między nimi jest oczywiście współdzielona. Podzielone zostały następująco:

- EasyRent.Identity – Moduł służy do uwierzytelniania i autoryzacji użytkowników. Oprócz tego zawiera metody odpowiedzialne za tworzenie i edycje kont. Część jego danych jest synchronizowana wraz z innymi modułami
- EasyRent.Management – Odpowiada za funkcję moderacji użytkowników przez administratora systemu
- EasyRent.RentalAd – Główny moduł aplikacji. Zawiera logikę związaną z tworzeniem i edycją ogłoszeń wynajmu.
- EasyRent.Reservation – Posiada logikę służącą do tworzenia i zarządzania rezerwacjami najmów. Synchronizuje dane bezpośrednio z modułami RentalAd oraz Identity



Rysunek 16. Relacje zachodzące między modułami

5.2. Warstwy pojedynczego modułu

Każdy z modułów składa się z kilku warstw. Cztery pierwsze warstwy opisane poniżej używane są w każdym module, a następne trzy wykorzystywane są w zależności od potrzeb. Opis warstw modułu:

- EasyRent.{Nazwa modułu} – zawiera kontrolery, pliki JSON z konfiguracją aplikacji, modele żądań (z ang. requests) API
- EasyRent.{Nazwa modułu}.Core – zawiera elementy niezbędne do obsługi poleceń i zapytań wdług wzorca CQRS. Oprócz tego są tutaj zawarte interfejsy niezbędnych do wywoływania modelów biznesowych. Sama warstwa nie zna logiki biznesowej, bardziej skupia się na poprawnym wywołaniu warstwy domeny
- EasyRent.{Nazwa modułu}.Domain – Centrum każdego modułu, tutaj znajdują się wszystkie agregaty, encje oraz obiekty wartościowe. Zawarta tutaj logika oraz reguły biznesowe reprezentują wymagania funkcjonalności dla klienta. Zawarte są tutaj także zdarzenia związane z Event sourcingiem, nazywane zdarzeniami domenowymi oraz interfejsy serwisów domenowych, których implementacja znajduję się w warstwie infrastruktury. Dzięki temu warstwa ta nie jest sprzężona z żadnymi zewnętrznymi zasobami
- EasyRent.{Nazwa modułu}.Infrastructure – Implementuje logikę dla wszystkich abstrakcji utworzonych w innych serwisach. Zawarta została tutaj implementacja klas służących do łączenia się z bazami danych oraz logika projekcji zdarzeń domenowych. Moduł odwołuje się do większości zewnętrznych zasobów, wyjątkiem jest np. Biblioteka MediatR która została wykorzystana do

implementacji wzorca CQRS w warstwie Core. Znajduję się tutaj także logika konsumująca zdarzenia integracyjne służące do synchronizacji danych między modułami z wykorzystaniem brokera RabbitMQ w sposób asynchroniczny oraz implementacja fasad dzięki którymi możliwe jest synchroniczne pobranie danych z innego modułu

- EasyRent.{Nazwa modułu}.Integrations – Składa się z logiki służącej do generowania zdarzeń integracyjnych
- EasyRent.{Nazwa modułu}.ReadModels –W tym module znajdują się niemodyfikowalne modele służące tylko do odczytu. Warstwa ta może być wykorzystywana przez inne moduły
- EasyRent.{Nazwa modułu}.Shared – Służy do udostępniania abstrakcji fasady dostępu do tego modułu wraz z odpowiednimi metodami. W niektórych przypadkach znajdują się tu, także stałe, które są wykorzystywane przez inne moduły jednak logicznie są związane z modułem macierzystym (np.: polityki uwierzytelniania (z ang. *Authorization policies*))

5.3. Wybrane biblioteki zewnętrzne

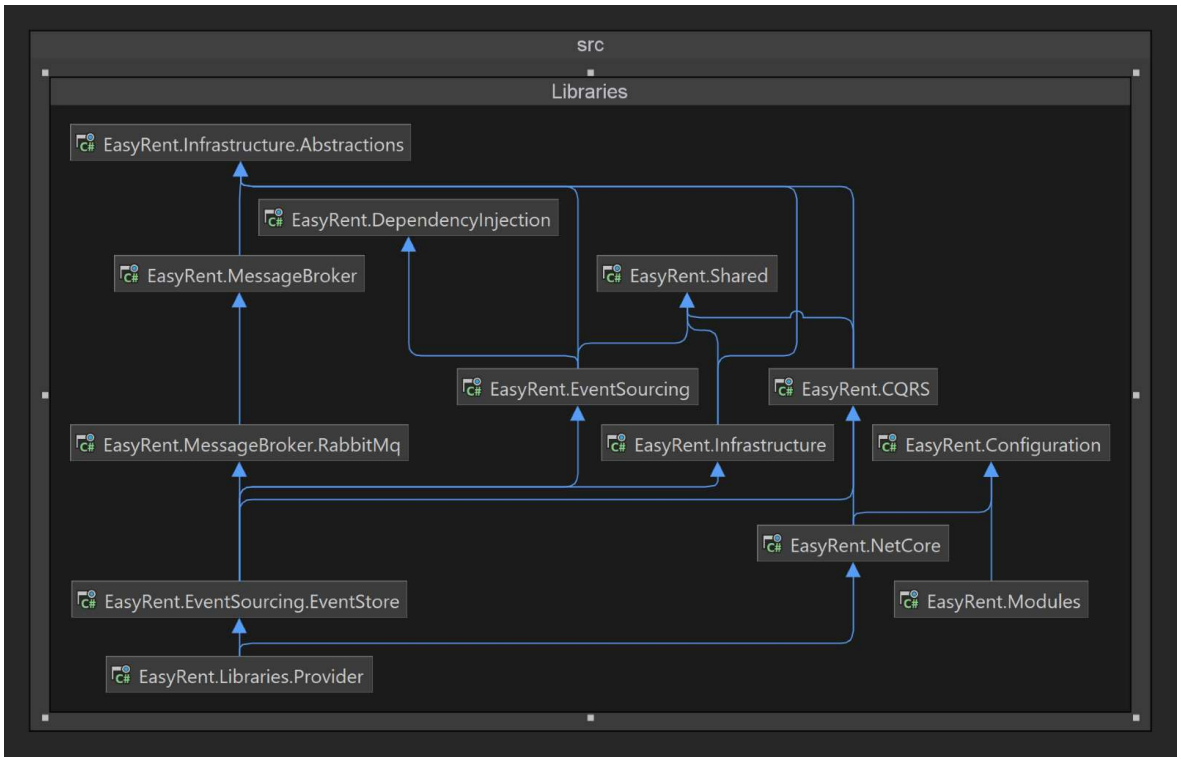
W ramach projektu wykorzystane zostały popularne biblioteki dostępne na rynku, takie jak:

- MediatR – Biblioteka służąca do prostej implementacji wzorca CQRS
- Fluent Validations – Służy do walidacji zapytań API
- Newtonsoft.Json – Biblioteka do serializacji i deserializacji danych
- NLog – Biblioteka służąca do logowania występujących w aplikacji komunikatów
- AutoMapper – Dzięki tej bibliotece można w łatwy sposób mapować obiekty między sobą

5.4. Biblioteki wewnętrzne

Poszczególne moduły aplikacji korzystają ze współdzielonego kodu, który rzadko będzie się zmieniał. W ramach tego wydzielone zostały odpowiednie biblioteki wewnętrzne:

- EasyRent.Configuration – zawiera klasy związane z ogólną konfiguracją modułów
- EasyRent.CQRS – ta biblioteka posiada metody rozszerzające wykorzystanie biblioteki mediatr
- EasyRent.DependencyInjection – tutaj znajdują się funkcję do zarządzania kontenerami IoC
- EasyRent.EventSourcing – zawiera metody i klasy ogólnie związane z Event Sourcingiem
- EasyRent.EventSourcing.EventStore – w tym miejscu znajdują się klasy i metody bezpośrednio związane z magazynem wydarzeń Event Store
- EasyRent.Infrastructure – tutaj zawarte są klasy związane z bazą danych MongoDB. Znajduję się tutaj, także implementacja serwisu do zapisu plików w chmurze Cloudinary
- EasyRent.Infrastructure.Abstractions – Biblioteka zawiera abstrakcję dla biblioteki wymienionej powyżej
- EasyRent.Libraries.Provider – tutaj znajduję się metoda służąca do wstrzykiwania wszystkich kontenerów IoC z wszystkich wymienionych bibliotek
- EasyRent.MessageBroker – składa się z ogólnych klas dla message brokera
- EasyRent.MessageBroker.RabbitMQ – znajdują się tutaj klasy potrzebne do implementacji biblioteki RabbitMQ
- EasyRent.Modules – posiada klasy związane z modułami
- EasyRent.NetCore – tutaj zawarte są klasy typowo związane z .NET Core takie jak: podstawowe klasy kontrolerów, corsy, filtry i middlewary
- EasyRent.Shared – zawiera modele wykorzystywane przez wszystkie moduły, klasy dla paginacji, wyjątki oraz metody rozszerzeń (z ang. *Extension methods*)



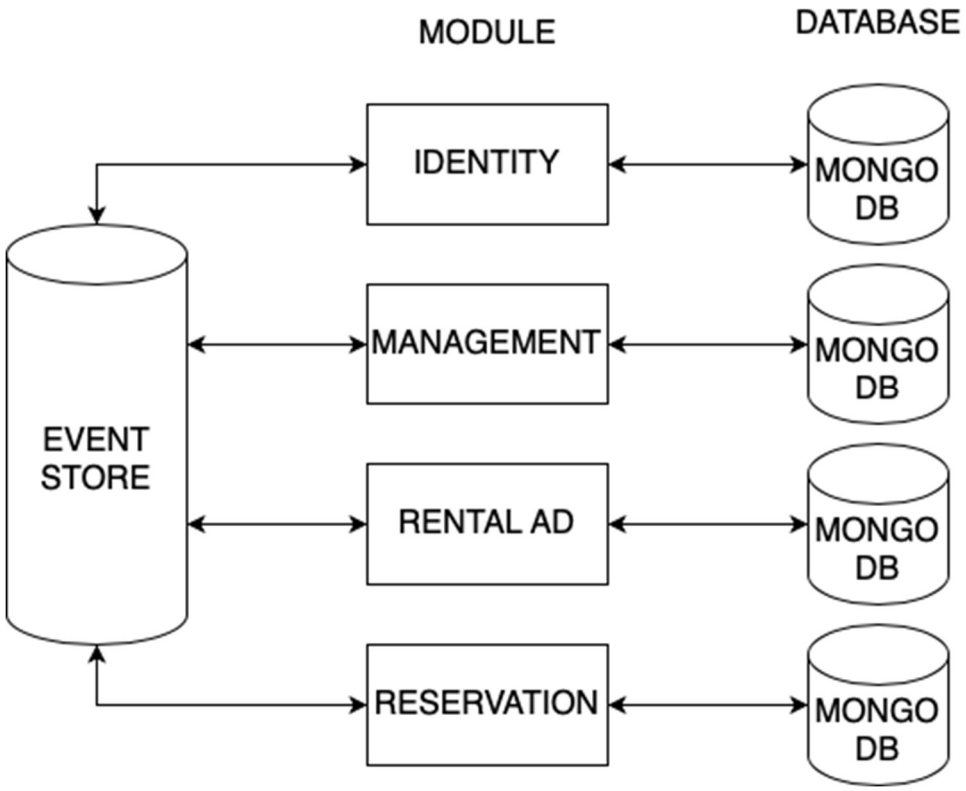
Rysunek 17. Model relacji między bibliotekami

5.5. Opis baz danych

Wszystkie zdarzenia zmieniające stan encji zachowywane są w strumieniu magazynu zdarzeń. Oprócz tego każdy moduł posiada dedykowaną sobie bazę danych, do których dane zapisują projekcje (zagadnienie to zostało szerzej opisane w rozdziale 5.7) wywoływane wraz z odpowiednim zdarzeniem (patrz Rys. 18.). W tych bazach przechowywane są tylko aktualne stany modeli domenowych. Z racji tego, że bazy te służą głownie do odczytu wybór padł na dokumentową bazę danych, gdyż ich wydajność jest większa od klasycznych baz relacyjnych.

Zapisywanie danych jako dokumentu sprawia, że są one bliższe realnemu modelowi używanemu w aplikacji oraz ich schemat jest definiowany przy odczycie (z ang. *Schema on read*) co sprawia, że wykorzystanie ich będzie sprzyjało dalszemu rozwojowi aplikacji. Mimo wszystko warto wspomnieć i opisać pozostałe typy baz nierelacyjnych. Bazy grafowe świetnie sprawdzają się w wypadku dużej ilości relacji między obiektami szczególnie tymi many–to–many. Niestety w wypadku tego projektu takie rozwiązanie nie ma prawa bytu, ponieważ tych relacji jest niewystarczająca ilość. Kolejnym typem jest baza zorientowana kolumnowo. Plusem takiej bazy jest możliwość manipulacji kolumnami, tak by przy zapytaniu niechciane dane nie używały dodatkowo pamięci. To

rozwiązanie sprawdza się do celów analitycznych w przypadku wielkich zbiorów danych. Ostatnim rodzajem jest klucz–wartość, z racji braku podobnych struktur danych w projekcie, ta baza również nie znalazła zastosowania [20, 21, 22].



Rysunek 18. Model relacji między modułami, a bazami danych

5.6. Bezpieczeństwo

Logowanie na konto użytkownika odbywa się za pomocą hasła. Jest ono hashowane algorytmem Argon2Id z wykorzystaniem soli (z ang. *Password Salt*). Algorytm ten jest rekomendowany przez społeczność OWASP oraz jest on zwycięzcą PHC organizowanego w 2015 roku [23]. Jego konfiguracja została ustawiona zgodnie z minimalnymi wytycznymi rekomendowanymi przez wyżej wspomnianą społeczność [24].

W celach autoryzacji użytkowników wykorzystane zostało rozwiązanie oparte na JWT – JSON Web Token. Jest on zaszyfrowany z wykorzystaniem algorytmu haszującego sha256. W tokenie zawarte są informacje na temat aktualnie zalogowanego – jego identyfikatora, maila, statusu i typu konta. Dzięki temu, poszczególne endpointy mogą być używane tylko przez odpowiednie typy kont albo tylko dla zalogowanych użytkowników. Token przesyłany jest w nagłówku każdego zapytania API. Ma on ważność 24 godzin, a po tym okresie musi zostać wygenerowany na nowo za pomocą logowania. Użytkownik,

który nie posiada dostępu do danego zasobu, po próbie wywołania danego zasobu, zostanie przekierowany na stronę główną aplikacji. Przy bezpośredniej próbie wywołania zasobu z API aplikacji (np.: z użyciem skryptu curl), zostanie zwrócony odpowiedni błąd [25].

5.7. System projekcji danych

Zadaniem projekcji jest odzwierciedlać zdarzenia, które miały miejsce, w bazie przeznaczonej do odczytu. By to osiągnąć, w oprogramowaniu działa specjalny serwis, którego zadaniem jest nasłuchiwanie strumienia zdarzeń znajdującego się w Event Store. W momencie dostarczenia zdarzenia do strumienia, zdarzenie wraz z zawartymi w nim danymi jest przechwytywane przez serwis i wywołana jest odpowiednia projekcja. Jako przykład wykorzystane zostało zdarzenie *PlaceReservationCreated* (patrz Rys. 19.). Gdy zostanie ono dodane do strumienia w Event Store, uruchomiona zostanie projekcja *PlaceReservationCreatedProjection* (patrz Rys. 20.). Zadaniem tej projekcji będzie sprawdzenie, czy podany obiekt został już utworzony w bazie. Gdy obiekt nie istnieje w bazie danych, projekcja utworzy obiekt wraz z danymi zawartymi w zdarzeniu.

```
public record PlaceReservationCreated(
    Guid PlaceReservationId,
    DateTime ArrivalDate,
    DateTime DepartureDate,
    Guid TenantId,
    Guid RentalAdId
) : IDomainEvent;
```

Rysunek 19. Model zdarzenia utworzenia rezerwacji

```
public class PlaceReservationCreatedProjection : DefaultProjection<PlaceReservationCreated>
{
    filipoficialnie
    public PlaceReservationCreatedProjection(ILogger logger, IDIProvider diProvider) : base(logger, diProvider)
    {
    }

    0+1 usages  filipoficialnie +1 *
    protected override async Task ProjectEventAsync(PlaceReservationCreated @event)
    {
        var documentRepository = Scope.ResolveService<IPlaceReservationDocumentRepository>();

        if (!await documentRepository.ExistsAsync(pr :PlaceReservationDocument
                => pr.PlaceReservationId == @event.PlaceReservationId))
        {
            await documentRepository.StoreAsync(new PlaceReservationDocument
            {
                PlaceReservationId = @event.PlaceReservationId,
                ArrivalDate = @event.ArrivalDate,
                DepartureDate = @event.DepartureDate,
                TenantId = @event.TenantId,
                RentalAdId = @event.RentalAdId,
                State = PlaceReservationState.Ongoing.ToString()
            }); // Task

            Logger.Trace( message: "> Place reservation with ID: #{ReservationId} stored in the document store",
                @event.PlaceReservationId);
        }
    }
}
```

Rysunek 20. Klasa przedstawiająca logikę projekcji zdarzenia utworzenia rezerwacji

5.8. Synchronizacja danych między modułami

Do synchronizacji danych wykorzystywane są zdarzenia integracyjne oraz kolejka RabbitMQ. Jeden z modułów nadaje zdarzenie do kolejki RabbitMQ z pomocą serwisu do publikowania zdarzeń integracyjnych. Gdy zdarzenie jest już w kolejce, zostaje ono skonsumowane przez działający w tle serwis (z ang. *Background Service*). W zależności od serwisy z którego event został nadany, serwis uruchomi odpowiednie klasy konsumujące (z ang. Consumer) (rys. 21, 22). Przykładem będzie wywołanie zdarzenia *ReservationCreatedIntegrationEvent.* Zdarzenie to publikowane jest w *CreatePlaceReservationCommandHandler* w module *Reservation.* Następnie jest konsumowany w module *RentalAd.* W rezultacie czego wywoływane jest polecenie (z ang. Command) *AddRentalAdReservationCommand.*

```
protected override async Task<bool> OnMessageReceivedAsync(ReservationIntegrationEvent message,
    CancellationToken cancellationToken)
{
    using (var scope = _diProvider.CreateScope())
    {
        _mediator = scope.ResolveService<IMediator>();

        Logger.Info(message: "Reservation event type: {EventType}", message.EventType);

        var command:IBaseRequest = PrepareReservationCommand(message);

        if (command is null)
        {
            Logger.Trace(message: "Reservation integration event does not match to any command");
            return true;
        }

        var result:object = await _mediator.Send(command, cancellationToken);

        return ((BaseResponse)result).IsSucceeded;
    }
}
```

Rysunek 21. Metoda, która jest wywoływana po otrzymaniu odpowiedniego zdarzenia

```
private static IBaseRequest PrepareReservationCommand(ReservationIntegrationEvent message)
    => message.EventType switch
    {
        ReservationEventType.ReservationCanceled => new RemoveRentalAdReservationCommand
        {
            RentalAdId = message.RentalAdId,
            RentalAdReservationId = message.RentalAdId
        },
        ReservationEventType.ReservationCreated => PrepareAddRentalAdReservationCommand(message),
        _ => null
    };
```

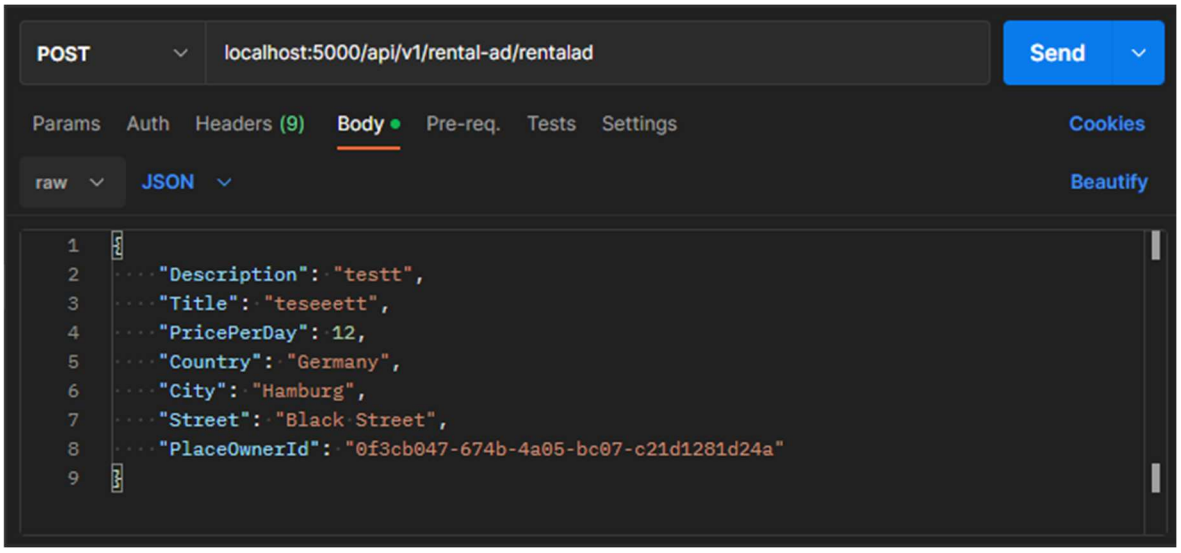
Rysunek 22. Metoda mapująca zdarzenia integracyjne do odpowiednich poleceń

Rozdział 6
Weryfikacja i walidacja

6.1. Testowanie

Wszystkie scenariusze zostały przetestowane manualnie. Sprawdzona została poprawność oraz długość wpisywania danych, a także ich format. Na etapie tworzenia części backend-owej wszystkie przygotowane endpointy zostały przetestowane ręcznie z wykorzystaniem aplikacji P*ostman* (rys. 1.), która służy do wykonywania zapytań API oraz automatyzacji procesu testowania. Oprócz tego część logiki biznesowej, zawartej w warstwie domeny została przetestowana za pomocą testów jednostkowych z wykorzystaniem biblioteki *Xunit.* Miały one na celu sprawdzenie zachowania obiektów dziedziny w przypadku próby interakcji z nimi z wykorzystaniem błędnych danych. Przykładem może być test, który sprawdza czy zwrócony zostanie wyjątek w wypadku braku podania identyfikatora użytkownika tworzącego ogłoszenie (patrz Rys. 2.). Warto zwrócić uwagę na konwencję nazewnictwa wykorzystaną w przypadku tych testów. Zawiera ona dokładną nazwę testowanej metody, scenariusz oraz oczekiwany rezultat. Sam test został napisany wykorzystując wzorzec AAA (Arrange, Act, Assert), co zostało nawet za komentowane w kodzie by uwypuklić poszczególne elementy tego wzorca [26].

Strona kliencka została także porównana z wytycznymi WCAG 2.1 [27]. W rezultacie czego okazało się, że spełnia ona większość zawartych tam reguł. Aplikacja jest walidowana głównie na poziomie backendu, aczkolwiek w warstwie klienta istnieją komunikaty mówiące o błędach w formularzach. Zbiór reguł walidacji dla każdego modułu zawarty jest w klasie: {Nazwa Modułu}ValidationRules (patrz. Rys. 3.). Umożliwia to zmianę parametrów w całej aplikacji z poziomu jednego miejsca.



Rysunek 23. Testowe zapytanie przygotowane w aplikacji Postman

```
[Fact]
filippyziak
public void CreateRentalAd_WhenPlaceOwnerIdIsEmpty_ShouldThrowArgumentException()
{
    //Arrange
    var invalidPlaceOwnerId = Guid.Empty;

    //Act
    Action act = () => new Domain.RentalAd.RentalAd(id:new RentalAdId(RentalAdId),
        new PlaceOwnerId(invalidPlaceOwnerId),
        PlaceOwnerEmailAddress.FromString(StubEmailAddress),
        RentalAdTitle.FromString(StubText),
        RentalAdDescription.FromString(StubText),
        new PlaceAddressId(PlaceAddressId),
        PlaceAddressCountry.FromString(StubCountry),
        PlaceAddressCity.FromString(StubCity),
        PlaceAddressStreet.FromString(StubStreet),
        RentalAdPricePerDay.FromDecimal(StubPricePerDay));

    //Assert
    act.Should().Throw<ArgumentException>();
}
```

Rysunek 24. Kod testu jednostkowego sprawdzający poprawność walidacji przy tworzeniu ogłoszenia najmu

```
public static class PlaceReservationValidationRules
{
    6 usages  filippyziak
    public static class Score
    {
        public const int MinAverageScore = 1;
        public const int MaxAverageScore = 5;
    }

    6 usages  filippyziak
    public static class ReservationReview
    {
        public const int MinLength = 1;
        public const int MaxLength = 500;
    }
```

Rysunek 25. Plik zawierający wartości do walidacji rezerwacji

6.2. Wykryte i usunięte błędy

Podczas pracy nad projektem napotkane zostały różne błędy. Niepoprawne działanie filtra wolnych terminów – błąd w zapytaniu. Początkowe nierozpatrzenie wszystkich przypadków, spowodowało możliwość rezerwowania zajętych już terminów. Błędnym przypadkiem była np.: próba rezerwacji terminu od 10 do 17 grudnia, gdy 13 do 15 grudnia był już zarezerwowany. Błąd ten uniemożliwiał, także poprawne wyszukiwanie ogłoszeń.

Dużym błędem było także wykorzystanie PostreSQL do relacji dla samych adresów, gdyż są one i tak zapisywane w dokumencie ogłoszenia w bazie danych. Dodawało to niepotrzebną złożoność systemu oraz spowalniało działanie aplikacji. Natomiast wynikało to z braku wiedzy o tym, że w bazie danych MongoDB mogą być tworzone indeksy na zagnieżdżonych obiektach.

Wystąpiły również problemy sieciowe związane z kontenerami Docker. Przy pierwszych uruchomieniach aplikacja niepoprawnie łączyła się z kontenerami zawierającymi bazy danych. Rozwiązaniem tego problemu było poprawienie składni pliku konfiguracyjnego *docker–compose.yml*.

Rozdział 7
Podsumowanie i wnioski

7.1. Perspektywy rozwoju

W świetle wymienionych w rozdziale trzecim wymagań w procesie tworzenia niniejszego projektu udało zrealizować się wszystkie postawione założenia.

Ważnym aspektem rozwoju jest kliencka strona aplikacji. Jedną z perspektyw stanowi dostosowanie interfejsu do wytycznych dotyczących dostępności – WCAG 2.1 (Web Content Accessibility Guidelines), opierających się na zasadach postrzegalności, funkcjonalności, zrozumiałości i solidności [27]. Aplikacja może zostać rozbudowana o szereg funkcjonalności, w tym wbudowanie systemu płatności w aplikację czy usprawnienie autoryzacji użytkownika w taki sposób, aby tokeny JWT miały krótszą ważność i odświeżały się automatycznie. W serwisie w zamyśle skonstruowanym jako platforma pośrednicząca między dostawcami a odbiorcami usługi dobrym pomysłem wydaje się dodanie czatu. Umożliwiłoby to sprawniejszą komunikację między twórcami ogłoszeń a osobami zainteresowanymi najmem. Sama wyszukiwarka ogłoszeń mogłaby ponadto zostać wzbogacona o mechanizm sugerujący użytkownikom lokale w oparciu o ich preferencje, tj. narzędzie filtrujące. W przypadku pojawienia się dużej ilości nowych użytkowników i zwiększenia obciążenia serwerów do maksimum, zadowalającym rozwiązaniem mogłoby okazać się odejście od architektury modularnego monolitu i zastąpienie jej architekturą mikrousług.

Dla jak największej niwelacji występowania niechcianych błędów związanych z przyszłymi planami rozwoju aplikacji odpowiednie byłoby dodanie brakujących testów jednostkowych, tak by pokrywały całą logikę biznesową. W celach uzyskania jeszcze lepszej ochrony przed pojawieniem się nieprawidłowości dobrym rozwiązaniem byłoby wprowadzenie innych rodzajów testów automatycznych takich jak – testy integracyjne lub testy komponentów.

7.2. Wnioski

W niniejszym projekcie wykorzystano wielorakie zagadnienia i rozwiązania techniczne, w tym architekturę modularnego monolitu, *Event Sourcing*, *CQRS* oraz podejście *Domain-Driven Design*. Pomimo że ułatwiają one skalowalność aplikacji oraz jej późniejszy rozwój, wymagają większego nakładu pracy aniżeli klasyczne rozwiązania. Rozważając perspektywę biznesową zastosowanych technologii, należy jednak zaznaczyć, że wielu potencjalnych klientów tych rozwiązań może w ogóle nie być nimi zainteresowana, zwracając uwagę na kosztowność rozwoju systemu na najwcześniejszym jego etapie, oraz późniejsze utrzymanie.

Jak w innych naukach tak i w sferze informatycznej wybrzmiewa konieczność dostosowania narzędzi do wymogów interdyscyplinarności. W skutek postępującej specjalizacji zaburzony został balans pomiędzy pogłębianiem wiedzy w zakresie jednej dziedziny, a bardziej holistycznym spojrzeniem. Konieczne jest, aby programiści nie tylko udoskonalali swój warsztat, ale wzbogacali swoją wiedzę o zrozumienie struktur biznesowych – tak, żeby lepiej i świadomiej dobierać rozwiązania do wymagań rynku. Wydaje się również, że tematyka systemów rozproszonych, architektury modularnego monolitu, *Domain-Driven Design* i *Event Sourcing* – jako technologii cieszących się dużym zainteresowaniem wśród programistów – zasługują na uwzględnienie w programach uczelni wyższych w Polsce, co mogłoby przełożyć się na większą konkurencyjność krajowych specjalistów w przyszłości.

Bibliografia

[1] Ashley Lightfoot, *How Booking.com Became Travel's Biggest Brand*, "Latana", 2022, online: https://latana.com/post/booking–deep–dive (dostęp: 11.01.2023)

[2] David Curry, *Airbnb Revenue and Usage Statistics*, "Business of Apps",2023, online: https://www.businessofapps.com/data/airbnb–statistics/ (dostęp: 11.01.2023)

[3] *Airbnb fourth quarter and full–year 2021 financial results*, "Airbnb", 2022, online: https://news.airbnb.com/airbnb–fourth–quarter–and–full–year–2021–financial–results/ (dostęp: 11.01.2023)

[4] Sam Newman, *Budowanie mikrousług. Szybkie wprowadzenie. Wykorzystaj potencjał architektury usług!,* Helion, Gliwice 2015, s. 20–29

[5] Alexey Zimarev, *Domain–Driven Design dla .NET Core*, Helion, Gliwice 2021, s. 359–363

[6] Miłosz Lenczewski, *What is better? Modular Monolith vs Microservices*, "Medium", 2021, online: https://medium.com/codex/what-is-better-modular-monolith-vs-microservices-994e1ec70994 (dostęp: 11.01.2023)

[7] Eric Evans, *Domain–Driven Design. Tackling Complexity in the Heart of Software*. Bell & Bain Ltd, Glasgow 2004

[8] *Entities, Value Objects, Aggregates and Roots*, "Se Habla Code los Techies", 2008, online: https://lostechies.com/jimmybogard/2008/05/21/entities–value–objects–aggregates–and–roots (dostęp: 11.01.2023)

[9] Pablo Martinez, *Domain–Driven Design: Everything You Always Wanted to Know About it, But Were Afraid to Ask*, "Medium", 2020, online: https://medium.com/ssense–tech/domain–driven–design–everything–you–always–wanted–to–know–about–it–but–were–afraid–to–ask–a85e7b74497a (dostęp: 11.01.2023)

[10] Laurent Grima, *An introduction to Domain–Driven Design*, "Medium", 2018, online: https://medium.com/inato/an–introduction–to–domain–driven–design–386754392465 (dostęp: 11.01.2023)

[11] Eric Evans, *Domain–Driven Design. Tackling Complexity in the Heart of Software*. Bell & Bain Ltd, Glasgow 2004, s.24–27, 89–93, s. 125–135

[12] Greg Young, *CQRS Documents by Greg Young*, 2010, online: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (dostęp: 11.01.2023)

[13] Amanda Bennett, *Introduction to CQRS*, "Medium", 2021, online: https://medium.com/microservicegeeks/introduction–to–cqrs–64f609544f4a (dostęp: 11.01.2023)

[14] Alexey Zimarev, *Domain–Driven Design dla .NET Core*, Helion, Gliwice 2021, s. 234–252

[15] Brent Roose, *Starting with event sourcing*, 2021, online: https://stitcher.io/blog/what-event-sourcing-is-not-about (dostęp: 11.01.2023)

[16] Alexey Zimarev, *Domain–Driven Design dla .NET Core*, Helion, Gliwice 2021, s. 278–279

[17] Martin Kleppmann, *Designing Data–Intensive Applications. The Big Ideas Behind Reliable, Scalable and Maintainable Systems,* O'Reilly, Sebastopol 2017, s. 162–164

[18] Kamil Grzybek, *Modular Monolith: Domain–Centric Design*, 2020, online:

[19] *Will Monolith Replace Microservices Architecture?,* "Medium", 2022, online: https://medium.com/att–israel/will–modular–monolith–replace–microservices–architecture–a`8356674e2ea (dostęp: 11.01.2023)

[20] *Understanding the Different Types of NoSQL Databases*, "MongoDb", 2023, online: https://www.mongodb.com/scale/types–of–nosql–databases (dostęp: 11.01.2023)

[21] *When to Use a NoSQL Database*, "MongoDb", 2023, online https://www.mongodb.com/scale/when–to–use–nosql–database (dostęp: 11.01.2023)

[22] Hitesh Mishra, *SQL Vs NoSQL Database,* "Medium", 2021, online: https://medium.com/geekculture/sql–vs–nosql–database–c6bbfc73db54 (dostęp: 11.01.2023)

[23] 2019, online: https://www.password–hashing.net/#phc (dostęp: 11.01.2023)

[24] *Password Storage Cheat Sheet*, "OWASP", 2021, online: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (dostęp: 11.01.2023)

[25] Prabath Siriwardena, *JWT Internals and Applications*," Medium", 2020, online: https://medium.facilelogin.com/jwt–internals–and–applications–d6171ce9e9ce (dostęp: 11.01.2023)

[26] Paulo Gomes, *Unit Testing and the Arrange, Act and Assert (AAA) Pattern*, ”Medium”, 2017, online: https://medium.com/@pjbgf/title–testing–code–ocd–and–the–aaa–pattern–df453975ab80 (dostęp: 11.01.2023)

[27] WCAG 2.1 w skrócie, “Strona rządowa państwa polskiego”, 2021, online: https://www.gov.pl/web/dostepnosc–cyfrowa/wcag–21–w–skrocie (dostęp: 11.01.2023)

Spis skrótów i symboli

DDD Domain–Driven Design, metodologia pracy przy projekcie programistycznym

OWASP Open Web Application Security Project, społeczność zajmująca się tematem bezpieczeństwa internetowego

AAA Arrange, Act and Assert Pattern, wzorzec wykorzystywany do testów jednostkowych

JWT Json Web Tokens, token słuzący do autoryzacji

CQRS Command Query Responsibility Segregation – wzorzec projektowy

JPG Joint Photographic Experts Group, format plików graficznych

PNG Portable Network Graphics, format plików graficznych

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

- źródła programu.

Spis rysunków