Złożenie pracy online:
2021-03-28 11:57:50
Kod pracy:
8263/38361/CloudA

Adam Wróbel
(nr albumu: 23584)

Praca inżynierska

Housematica - aplikacja do konfigurowania oraz zarządzania nieruchomościami

Housematica - application for configuring and managing real estate

Wydział: Wydział Nauk Społecznych i Informatyki

Kierunek: Informatyka

Specjalność: programowanie aplikacji biznesowych

Promotor: dr Krzysztof Przybycień

Nowy Sacz, 2021

Serdecznie dziękuję mojemu promotorowi doktorowi Krzysztofowi Przybycieniowi za wyrozumiałość, pomoc przy realizacji projektu oraz cenne wskazówki, których udzielał mi podczas tworzenia niniejszej pracy inżynierskiej

Streszczenie

Według danych statystycznych opublikowanych przez Główny Urząd Statystyczny z roku na rok zwiększa się popyt na nieruchomości. W przeważającej ilości przypadków, kupujący po zakupie mieszkania przeprowadzają jego modyfikacje dodając lub usuwając ścianki działowe. Taka operacja wiąże się z wydłużonym czasem oczekiwania na zasiedlenie danej nieruchomości. Z myślą o takich osobach w ramach niniejszej pracy inżynierskiej powstała aplikacja Housematica, która pozwala deweloperowi w łatwy sposób stworzyć konfigurator mieszkania, aby kupujący już na etapie jego wyboru mógł wybrać interesujący go układ ścianek działowych czy nawet wyposażenie nieruchomości. Aplikacja H została podzielona na dwie części, pierwsza z nich - panel klienta pozwala za pomocą intuicyjnego interfejsu stworzyć konfigurację mieszkania, korzystając z wizualizacji modelu mieszkania w przestrzeni 3D. Druga część aplikacji to panel administratora, który dostarcza nie tylko prosty formularz do tworzenia konfiguratorów, ale także szereg funkcjonalności jak: tworzenie zespołów, przeglądanie statystyk konfiguracji oraz praca na licencji.

Słowa kluczowe

konfigurator nieruchomości, wizualizacje 3D, ASP.NET Core, wzorzec architektoniczny MVC, Babylon.js

Abstract

According to statistical data published by the Central Statistical Office, the demand for real estate is growing year by year. In the majority of cases, buyers, after purchasing an apartment, modify it by adding or removing partitions. Such an operation is associated with an extended waiting time for the occupation of a given property. With such people in mind, as part of this engineering thesis, the Housematica application was created, which allows the developer to easily create an apartment configurator, so that the buyer can choose the layout of partition walls or even the equipment of the property at the stage of his selection. The H application has been divided into two parts, the first one - the customer panel allows you to create an apartment configuration using an intuitive interface, using the visualization of the apartment model in 3D space. The second part of the application is the admin panel, which provides not only a simple form for creating configurators, but also a number of functionalities such as creating teams, viewing configuration statistics and working under a license.

Keywords

real estate configurator, 3D visualizations, ASP.NET Core, MVC architectural pattern, Babylon.js

SPIS TREŚCI:

Wstęp

W obecnych czasach rynek nieruchomości przeżywa prawdziwy „boom" zarówno pod względem ilości mieszkań wprowadzonych do sprzedaży, jak i sprzedanych nieruchomości (wzrost o 7,1% rok do roku[1] oraz o 25,3% względem ostatnich 10 lat[2]). Wraz z rozwojem rynku nieruchomości wzrastają także potrzeby nabywców nowych mieszkań i domów. Dlatego coraz częściej, obok atrakcyjnej ceny mieszkania, dobrej lokalizacji oraz odpowiedniego standardu lokum, stawiają oni na zakup mieszkania dostosowanego do ich potrzeb. Mając na myśli taki zwrot najczęściej mówi się o odpowiednim rozmieszczeniu pomieszczeń oraz ich wielkości. Dzisiaj klienci, kupując mieszkanie, stają przed możliwością wyboru mieszkań pod względem ściśle określonych konfiguracji (wskaźników): lokalizacji, ceny, metrażu mieszkania oraz pomieszczeń, a także rozmieszczenia i ilości pomieszczeń.

Lokalizacja mieszkania w danej ofercie to stała, na którą kupujący nie ma bezpośredniego wpływu, ponieważ fizycznie nie może przenieść ważącego 5 tysięcy ton budynku w miejsce odpowiadające mu lokalizacyjnie. Cenę można potraktować jako zmienną, którą w rozsądny sposób możemy negocjować, szukając satysfakcjonującego dla obu stron złotego środka. Warto jednak nadmienić, że negocjowana cena mieszkania – nawet jeśli developer trafi na klienta będącego mistrzem negocjacji – każdorazowo zamknie się w pewnych ramach. Dlatego wskaźnik ceny zawsze mieści się w pewnym (najczęściej ustalonym przez developera) przedziale. Kolejne niezmienne wskaźniki danej oferty mieszkania to jego metraż (oraz metraż pomieszczeń), ilość pomieszczeń oraz ich rozmieszczenie. Powierzchnia mieszkania to wskaźnik, który jest z wiadomych względów niezmienny, tak samo jak ilość pomieszczeń oraz ich metraż. Tylko w niewielkim odsetku mieszkań zdarza się, że developer dostosuje mieszkanie do naszych potrzeb, na przykład stawiając lub usuwając ścianki działowe. Taka operacja zwykle jest zlecana innym firmom podwykonawczym już po zakupie mieszkania. Jeśli zwrócimy uwagę, że takie firmy często posiadają swój kalendarz zleceń z pracami umówionymi na kilka tygodni/miesięcy w przyszłość, czas w którym klient będzie mógł wprowadzić się do zakupionego mieszkania drastycznie wzrasta. Co więcej, wynajęcie firmy podwykonawczej wiąże się także z dodatkowymi kosztami, ponieważ owa firma musi najpierw usunąć coś, co zrobił przed momentem developer i zrobić to na nowo.

[1] *Główny Urząd Statystyczny*. Odczytano 22 marca 2021, z https://stat.gov.pl/wyszukiwarka/?query=tag:transakcje+kupna+sprzeda%C5%BCy+nieruchomo%C5%9Bci
[2] *Customisation in practice: interactive product*, Odczytano 22 marca 2021, z https://programa.pl/en/2017/10/Customisation_in_practice_interactive_product_configurator

A gdyby tak, już na etapie projektu budowlanego, klient mógł wybrać jedną z kilku możliwych konfiguracji mieszkania? Jeśli założymy, że znaczny odsetek klientów dokonuje zakupu mieszkania zanim rozpocznie się ich budowa (sic!), a rozmieszczenie pomieszczeń (z wyłączeniem pomieszczeń, w których wymagana jest woda czy kanalizacja) na etapie projektu często może być dowolnie modyfikowane (ponieważ ściany wewnętrzne mogą być rozbierane i stawiane, a z takich najczęściej uformowane są pomieszczenia) – to czy klient już na etapie zakupu mieszkania od developera, które na razie jest tylko projektem na kartce, nie mógłby skonfigurować metrażu pomieszczeń, ich ilości oraz rozmieszczenia?

Na pierwszy rzut oka może wydawać się, że o ile klient na takiej operacji zyskuje wiele czasu oraz pieniędzy, to zalety, które niesie ze sobą takie rozwiązanie dla developera są nikłe. Patrząc na aktualną sytuację na rynku nieruchomości (a głównie na ogromny popyt Polaków na zakup własnego lokum) mieszkania prędzej czy później i tak zostaną sprzedane. Jednak każdy z deweloperów z pewnością bez zastanowienia stwierdzi, że najlepszą dla niego sytuacją jest sprzedaż jak największej ilości mieszkań jeszcze przed rozpoczęciem budowy. Natomiast umożliwienie przez developera – dostępnej tylko przed rozpoczęciem budowy – opcji dostosowania mieszkania do swoich potrzeb zmobilizuje kupujących do zakupu mieszkania wcześniej niż dotychczas.

Aby umożliwić zarówno developerom zarządzanie konfiguracjami oraz klientom tworzenie nowych konfiguracji, stworzona została webowa aplikacja Housematica, która w łatwy sposób pozwala obu stronom na korzystanie z dóbr konfiguracji mieszkań. Dzięki temu gotowemu rozwiązaniu developerzy mogą w szybki sposób skonfigurować swoją inwestycję oraz finalnie umieścić ją na własnej stronie internetowej. Jedynym ograniczeniem konfiguratora jest sama wyobraźnia developera. W standardowym wariancie pozwala on na skonfigurowanie z dostępnej puli układu pomieszczeń oraz ich metraży. Developer może w łatwy sposób dodać kolejne opcje do konfiguracji, między innymi stolarkę drzwiową i okienną, wykończenie pokoi, kuchni czy łazienki.

Z kolei znajdujący się po drugiej stronie „mieszkaniowej transakcji" klienci mogą skonfigurować mieszkania zgodnie z własnymi upodobaniami oraz zapisać swoje konfiguracje, aby w dowolnym czasie do nich powrócić. Platforma Housematica w łatwy sposób pozwala wybrać interesujące klienta mieszkanie, ale także dostosować rozmieszczenie pomieszczeń, styl mieszkania czy stolarkę okienną według własnego upodobania.

Housematica to także narzędzie pozwalające developerom nieruchomości na przeglądanie statystyk konfigurowanych nieruchomości oraz określenie teraźniejszych, jak i przyszłych potrzeb swoich klientów. Platforma w bardzo przystępny sposób pozwala na

zwiększenie przychodów związanych ze sprzedaży nieruchomości oferując już na etapie zamawiania nieruchomości dodatkowe pakiety wyposażenia.

W niniejszej pracy opisana została aplikacja webowa Housematica wraz z technologiami, które użyto do opracowania platformy. Praca porusza również wątki związane z rosnącym trendem konfiguracji przedmiotów. W ostatnim rozdziale przedstawiono wyniki testów jednostkowych oraz testów badających wydajność aplikacji. Dodatkowo pod adresem http://prezentacja.housematica.pl/ umieszczony został materiał wideo prezentujący zasadę działania, najważniejsze funkcje oraz możliwości aplikacji webowej Housematica.

1. Konfiguratory

Od lat najbardziej ekskluzywne marki takie jak Nike, Porsche, Adidas na największych rynkach świata umożliwiają konfigurację swoich produktów. Co więcej, trend kustomizacji produktów uznano za jeden z największych trendów zarówno w 2017, jak i 2018 roku. Można zapytać: skąd tak duża popularność konfiguracji produktów? Kolejny podrozdział spróbuje znaleźć odpowiedź na te nurtujące pytanie.

1.1.Psychologia działania konfiguratorów

Popularność konfiguratorów, które coraz śmielej pojawiają się i są obecne na największych zagranicznych, ale i coraz częściej polskich rynkach, tłumaczy słynna hipoteza znana jako „efekt posiadania”[3]. Eksperyment potwierdzający słuszność tej hipotezy opracował izraelski psycholog oraz zdobywca nagrody Nobla z Ekonomii Daniel Kahneman. W 2011 roku przeprowadził eksperyment, podczas którego podzielił osoby na trzy grupy. Pierwsza z nich miała postawić się w roli sprzedawców i wycenić kubek, który następnie mieli sprzedać. Druga z nich miała postawić się w roli kupujących i oszacować, ile mogliby zapłacić za kubek. Trzecia grupa miała wycenić kubek pod kątem jego otrzymania lub uzyskania za niego równowartości wypłaconej w gotówce. Wyniki badania były zaskakujące. Średnia wycena wartości kubka przez grupę pierwszą (sprzedawców) wynosiła około 27 zł, podczas gdy średnia cena zaproponowana przez kupujących kubek wynosiła niespełna 11 zł. Trzecia grupa wyceniła kubek na około 12 zł. Zastanawiające jest zatem skąd tak duża rozbieżność w cenie pomiędzy osobami, które kubek posiadały, a osobami, które chciały go nabyć.

Zasadniczo okazuje się, że posiadanie produktu, zanim klient za niego zapłaci, powoduje w mniemaniu klienta zawyżenie jego wartości oraz przywiązanie emocjonalne do niego. Zatem w jaki sposób można wykorzystać ten syndrom wśród osób zainteresowanych zakupem nieruchomości? Idealnym rozwiązaniem tej zagadki są konfiguratory nieruchomości, dzięki którym klienci dokonują decyzji o zakupie nieruchomości dopiero po przemyśleniu i

[3] Junk, D. (2017). *The 2 Psychological Principles that Make Product Configurators Such Stupidly Effective Ecommerce Tools*. Odczytano 22 marca 2021, z https://www.linkedin.com/pulse/2-psychological-principles-make-product-configurators-dennis-junk

przeanalizowaniu, jak domniemana nieruchomość ma wyglądać oraz jakie uczucia będzie wywoływać już sam fakt jej posiadania. Przykładowo: podczas konfigurowania koloru butów wyobrażamy sobie w jakich sytuacjach i do jakich ubrań będziemy nosić owe buty. W czasie konfigurowania samochodu wyobrażamy sobie jak jego systemy będą działać w momencie parkowania. W dokładnie taki sam sposób konfigurator nieruchomości pozwoli klientom na wyobrażanie sobie ich idealnego mieszkania jako własności oraz pozwoli na stworzenie więzi emocjonalnej z mieszkaniem. Każda zmiana w mieszkaniu w czasie konfiguracji zwiększa chęć posiadania go przez klienta. Eksperyment Daniela Kahneman'a dowodzi że można zwiększyć postrzeganą wartość produktów poprzez umożliwienie klientom wyobrażenia sobie, że są oni jego właścicielami. Możliwość dodania własnych akcentów do produktu jeszcze bardziej spotęguje ten efekt.

1.2 Znaczenie implementacji konfiguratorów nieruchomości w biznesie

Wyobraźmy sobie, że działamy na rynku, w którym istnieje 1000 firm z tej samej branży. Jak będzie wyglądała sytuacja, jeśli jedna z nich zaoferuje możliwość skonfigurowania swoich produktów, dzięki któremu odwiedzający stronę internetową będą mogli bawić się przy tworzeniu unikalnych konfiguracji nieruchomości lub domu? Po zakończeniu konfiguracji w swoim mniemaniu zainteresowany oceni swoją nieruchomość jako dwa razy bardziej wartościową niż tą, którą mogą przeglądać, jedynie w formie obrazka w galerii zdjęć, klienci na innej ze stron.

W momencie gdy klient będzie już fizycznym właścicielem swojego mieszkania, jego poziom satysfakcji z lokum wzrośnie, ponieważ towarzyszyć mu będzie przeświadczenie, że sam brał udział przy jego tworzeniu. Następnym razem, gdy znów będzie przymierzać się do ponownego zakupu nieruchomości, będzie pamiętać o konfiguratorze mieszkania jako przydatnej aplikacji i najpewniej po raz kolejny do niego wróci. Możliwe też, że poleci konfigurator swoim znajomym i rodzinie. Z upływem czasu możliwe jest, że konfiguracja mieszkania stanie się obowiązkowym elementem przy zakupie mieszkania i społeczeństwo niechętnie będzie wybierać nieruchomości od firm, które takiego rozwiązania nie oferują.

2. Cel i założenia

Głównym celem poniższego projektu jest stworzenie oprogramowania służącego do zaoferowania klientom developerów mieszkaniowych możliwości skonfigurowania sprzedawanych przez nich apartamentów. Na cel pracy składają się trzy główne założenia. Pierwsze założenie to stworzenie aplikacji webowej, w której potencjalny klient będzie mógł dokonać konfiguracji mieszkania poprzez wybranie danego apartamentu z bazy developera, skonfigurowania rozmieszczenia pomieszczeń oraz ewentualnego wyposażenia wnętrza. Aplikacja webowa zostanie wykonana zgodnie z założeniem „mobile first", w którym – jako pierwsze – projektujemy stronę pod kątem telefonów komórkowych. Aplikacja webowa przeznaczona dla klientów pozwoli również na wyświetlenie szacunkowego kosztu apartamentu, z wyszczególnieniem na wybrane składowe konfiguracji oraz na przegląd możliwości finansowania zakupu.

Drugie założenie aplikacji to stworzenie prostego i przejrzystego panelu administracyjnego, w którym developer będzie mógł w łatwy sposób zarządzać swoimi projektami oraz przeglądać statystyki dotyczące wyświetleń danego apartamentu czy stworzonych konfiguracji. Panel administracyjny przeznaczony dla developera umożliwi także kontakt z potencjalnymi klientami, którzy dokonali konfiguracji mieszkania. Co więcej, pozwoli również na tworzenie zespołów projektowych w ramach tych samych projektów. Dzięki takiemu rozwiązaniu, przy jednym projekcie będzie mogło pracować kilka osób, a dane go dotyczące będą mogły być udostępniane między nimi.

2.1. Użyte technologie – portal WWW

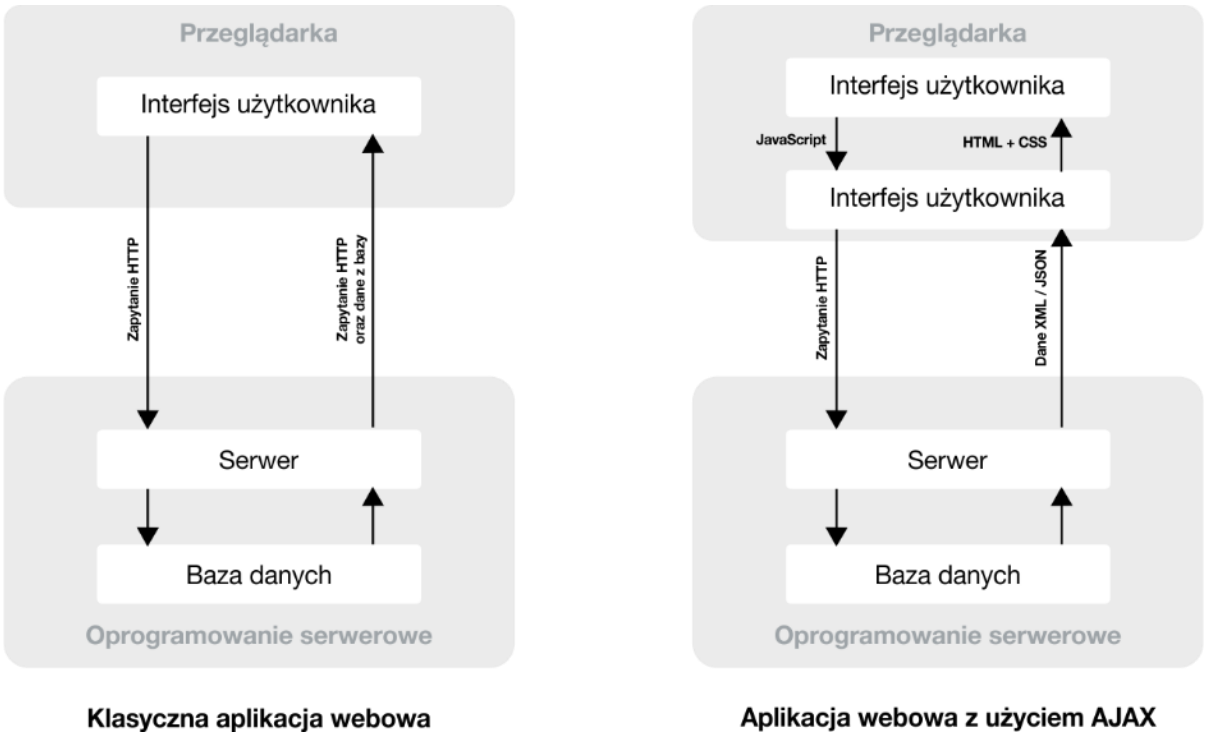
W ramach aplikacji webowej, w obrębie której klienci będą dokonywać konfiguracji oraz wyboru mieszkań, technologie dobrane zostały w taki sposób, aby zapewnić maksymalnie krótki czas dostępu do strony przy zachowaniu atrakcyjnego interfejsu użytkownika. Do zakodowania elementów strony użyty został hipertekstowy język znaczników HTML oraz kaskadowe arkusze stylów CSS. W celu łatwiejszego zarządzania kodem oraz strukturą strony użyta została biblioteka Bootstrap, która rozwijana jest przez twórców Twittera. Bootstrap to kolekcja wielokrotnego użytku gotowych komponentów kodu napisanych w HTML, CSS

i JavaScript, która umożliwia programistom tworzenie responsywnych stron internetowych[4]. Aby zapewnić stronie interaktywność, został użyty język programowania JavaScript, który plasuje się na 3. miejscu[5] pod względem popularności wśród języków programowania oraz na 1. miejscu[6] pod względem wielkości społeczności zgromadzonej według jednego języka. Zasadniczo JavaScript to skryptowy język programowania, który pozwala na implementację bardziej złożonych funkcji na stronach internetowych. Skrypty napisane w technologii JavaScript dostarczane są jako zwykły tekst, można napisać je bezpośrednio w kodzie HTML oraz uruchamiać bezpośrednio w przeglądarce. Nie potrzebują one specjalnego środowiska uruchomieniowego czy dodatkowej kompilacji do innego formatu. Co więcej, praca z JavaScript pozwala na użycie wielu bazujących na nim bibliotek. Przykładową biblioteką JavaScript, która użyta została w tym projekcie to jQuery, pozwalająca na łatwiejszą obsługę zdarzeń, wykonywanie animacji czy manipulowanie strukturą kodu HTML.

Kolejną często używaną biblioteką, która również znalazła zastosowanie w tym projekcie, jest biblioteka AJAX. AJAX (Asynchronous JavaScript and XML) to bardziej złożona biblioteka, która pozwala na asynchroniczne wysyłanie oraz odbieranie danych pomiędzy aplikacją a serwerem bez przeładowywania strony internetowej. Kiedy użytkownik ładuje stronę oraz przechodzi do akcji wywołującej zapytanie, AJAX JavaScript tworzy obiekt XMLHttpRequest, którego zadaniem jest przesłanie danych w formacie XML pomiędzy przeglądarką internetową a serwerem przetwarzającym żądania. Odpowiedź tworzona jest po stronie serwera oraz przesyłana z powrotem do przeglądarki, gdzie przetwarzana jest przy pomocy JavaScript oraz wyświetlana na ekranie. Sedno skuteczności technologii AJAX bazuje na braku przesyłania jeszcze raz tych samych danych (co ma miejsce przy odświeżaniu strony WWW), a skupieniu się tylko na dostarczeniu z serwera na stronę WWW porcji potrzebnych informacji.

[4] *Wikipedia.Wolna Encyklopedia.* Hasło: *Bootstrap (framework*). Odczytano 22 marca 2021, z https://pl.wikipedia.org/wiki/Bootstrap_(framework)
[5] *PopularitY of Programming Language*. Odczytano 22 marca 2021, z https://pypl.github.io/PYPL.html
[6] Tung. L. (2020). *Programming language popularity: JavaScript leads – 5 million new developers since 2017.* Odczytano 22 marca 2021, z https://www.zdnet.com/article/programming-language-popularity-javascript-leads-5-million-new-developers-since-2017/

Rys.2.1. Schemat działania technologii AJAX, opracowanie własne bazując na źródle [7]



Technologią użytą przy tworzeniu części konfigurowania mieszkań obsługującą zadania po stronie serwera jest ASP.NET w wersji 5. ASP.NET to technologia działająca po stronie serwera przeznaczona do tworzenia dynamicznych stron internetowych. Aplikacje w ASP.NET można pisać w wielu językach (np. C #, VB.Net i J#), mogą być one przeznaczone na komputery osobiste, telefony oraz przeglądarki internetowe. Ponadto, ASP.NET jest technologią wieloplatformową, dzięki czemu działa ona zarówno na urządzeniach wyposażonych w system operacyjny Windows, jak i Linux.

Ostatnią technologią jaka została użyta w projekcie portalu użytkownika jest silmnik renderujący 3D o nazwie Babylon.js wykorzystujący bibliotekę JasaScript. Opracowany został w 2015 roku przez pracowników firmy Microsoft Davida Catuhe oraz Davida Rousset. Babylon.js jest jednym z najpopularniejszych silników 3D rozwijany przy pomocy języka TypeScript skompilowanego do JavaScript, który oparty jest na WebGL. WebGL to API JavaScript, które jest natywnie interpretowane przez przeglądarki obsługujące HTML 5 i pozwala na renderowanie po stronie przeglądarki interaktywnych modeli 2D oraz 3D bez użycia dodatkowych wtyczek.

[7] *Institute of Computing Science, Poznan University of Technology AJAX w przykładach* odczytano dnia 20.03.2021, z http://www.cs.put.poznan.pl/jkobusinski/ajax.html

2.2. Użyte technologie – portal administracyjny

Technologiami odpowiedzialnymi za warstwę prezencji w części portalu administracyjnego jest HTML, CSS oraz Javascript. Ponadto w celu bardziej efektywnej pracy użyte zostały dodatkowe poniższe biblioteki/silniki:

- Handlebears – to bardzo popularny silnik tworzenia szablonów stron WWW. Jest prosty w użyciu oraz cechuje się dużą społecznością wykorzystujących go użytkowników. Dzięki silnikowi Handebears można oddzielić generowanie kodu HTML od reszty kodu JavaScript
  i pisać bardziej przejrzysty kod;
- jQuery – to biblioteka JavaScript odpowiedzialna za łatwiejszą obsługę zdarzeń, tworzenie animacji oraz manipulowanie strukturą HTML;
- SweetAlert2 – to biblioteka JavaScript pozwalająca na tworzenie w efektywny sposób przejrzystych komunikatów oraz alertów;
- Select2 – to biblioteka JavaScript pozwalająca na tworzenie konfigurowalnych pól wyboru, struktur wyświetlających dane oraz list typu Infininty Scroll;
- Moment.js – to biblioteka JavaScript pozwalające na bardziej efektywną pracę z obiektem JavaScript Date.

Po stronie serwera użyta została technologia ASP.NET, która przeznaczona jest do tworzenia dynamicznych stron internetowych. Dodatkowo, technologia ASP.NET wzbogacona została o mechanizm autoryzacji bazujący na mechanizmie ASP.NET Identity, który obsługuje zarządzanie dostępem do określonych zasobów aplikacji.

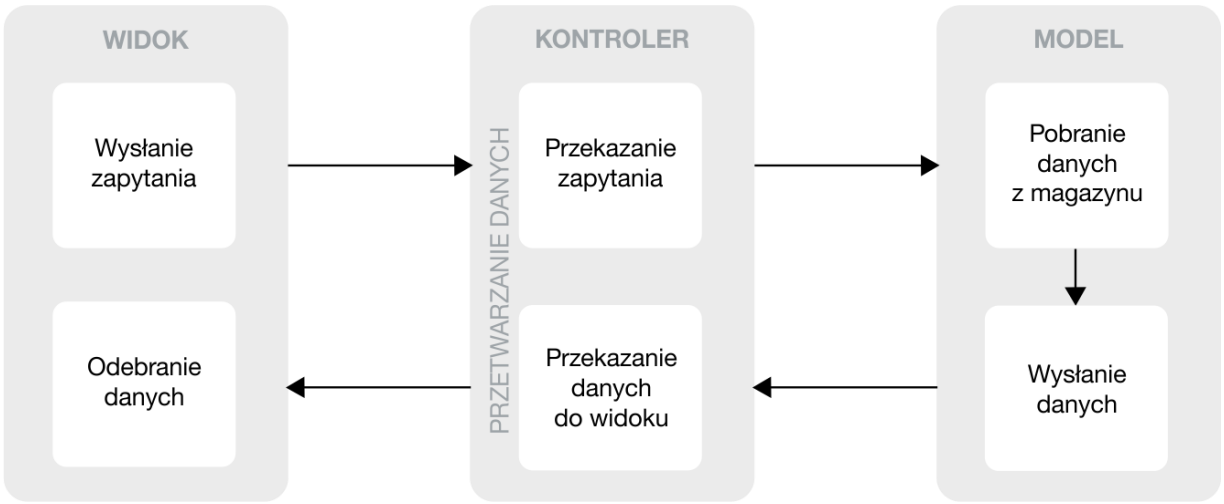
2.3. Wzorce architektoniczne

Wzorce architektoniczne to uniwersalne rozwiązanie problemów z zakresu architektury oprogramowania, która zawiera zasady dotyczące predefiniowanych podsystemów oraz zależności między nimi. W ramach projektu Housematica zarówno w wersji portalu klienta, jak i portalu administracyjnego użyty został wzorzec MVC.

Wzorzec MVC (ang. *Model – View – Controller*) to wzorzec architektoniczny, który rozdziela aplikację na trzy elementy: Model, Widok oraz Kontroler. Każdy z tych elementów posiada jasno zdefiniowane zadania, których obsługą ma się zająć. Wzorzec MVC jest jednym z najpopularniejszych wzorców projektowych stosowanym w projektach napisanych za pomocą różnych języków programowania (np. C#, Java, PHP). Dzięki jego użyciu

otrzymujemy schludnie stworzoną aplikację, podzieloną na niezależne od siebie komponenty, które możemy podmieniać[8]. Przykładowo, jeśli w wyniku rozwoju aplikacji dojdzie do konieczności zmiany w technologii dostarczania danych do aplikacji, można ją bez większych problemów zamienić bez modyfikowania większej ilości struktury kodu. Ponadto użycie wzorca MVC sprawdza się także w dużych projektach programistycznych, gdzie aplikację stworzoną we wzorcu MVC można łatwiej oraz bardziej przewidywalnie modyfikować. Ponadto architektura MVC ułatwia testowanie aplikacji, ponieważ dzięki podziałowi oprogramowania na osobne moduły można je bez trudu testować.

Rys.2.2. Schemat działania wzorca MVC, opracowanie własne bazując na źródle [9]



Rys.2.3. Zrzut ekranu przedstawiający przykładowy kontroler w aplikacji Housematica. Przedstawiony kontroler, oprócz zwracania widoku, realizuje dwie metody bazujące na poświadczeniach. Pierwsza z nich odpowiada za pobranie identyfikatora użytkownika, natomiast druga za pobranie identyfikatora licencji. Obie właściwości za pomocą obiektu ViewBag przekazywane są do widoku. Ponadto, w celach orientacyjnych, za pomocą klasy System.Diagnostic sprawdzany jest czas wykonania się bloku kodu.

```
[Authorize]
public IActionResult Index()
{

    var watch = System.Diagnostics.Stopwatch.StartNew();
    var claims = User.Identities.FirstOrDefault()?.Claims.FirstOrDefault()?.Value;
    var license = User.Identities.FirstOrDefault()?.Claims.LastOrDefault()?.Value;
    ViewBag.User = claims;
    ViewBag.License = license;
    watch.Stop();
    var elapsedTime = watch.ElapsedMilliseconds;


    return View();
}
```

[8] Adam Freeman *Pro (2020). ASP.NET Core 3 Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages Eight Edition* EAN: 9781484254394

[9] *Model–view–controller* odczytano dnia 20.03.2021, z *https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller*

Rys.2.4. Zrzut ekranu przedstawiający przykładowy model klasy bazodanowej w aplikacji Housematica. Klasa posiada pola, które przy pomocy Entity Framework zostaną odwzorowane jako kolumny tabeli w bazie danych. Ponadto w kodzie użyte zostały obiekty Data Annotations ([Key] oraz [ForeginKey]) które pozwolą na nadanie odpowiednim kolumnom klucza głównego oraz obcego.

```
[Key]
public Guid IdLicense { get; set; }
[ForeignKey("LicenseType")]
public int IdLicenseType { get; set; }
public DateTime Expire { get; set; }
public List<User> User { get; set; }
public List<ProjectsLicense> ProjectsLicense { get; set; }
public virtual LicenseType LicenseType { get; set; }
```

Rys.2.5. Zrzut ekranu przedstawiający przykładowy interfejs ILicense, który za pomocą mechanizmu wstrzykiwania zależności, zostanie użyty w kontrolerze. Interfejs posiada zbiór metod, których działanie zostanie napisane w klasie implementującej ten interfejs.

```
public interface ILicense
{
    public double percentageLicenseUsing(Guid License);
    public double projectAvailable(Guid License);
    public double userAvailable(Guid License);
    public double variantsAvailable(Guid License);
    public string licenseExpired(Guid License);
}
```

Rys.2.6 Zrzut ekranu przedstawiający przykładową implementację interfejsu ILicense. Metody umieszczone w klasie License będą odpowiadały za operacje dotyczące wyświetlania danych na widgetach dotyczących Licencji prezentowanych na stronie głównej aplikacji.

```
public class License : ILicense
{
    private readonly HousematicaContext _context;
    private List<Data.Data.CMS.License> license;
    public License(HousematicaContext context)
    {
        _context = context;
        license = _context.License.Include(x=>x.LicenseType).Include(x=>x.ProjectsLicense).ThenInclude(x=>x.Projects).ThenInclude
          (x=>x.Apartment).ToList();
    }

    public string licenseExpired(Guid License)
    {
        var licenseExpired = license.Where(x => x.IdLicense == License).Select(x => x.Expire).FirstOrDefault();
        return (licenseExpired.ToString("dd.MM.yyyy"));
    }

    public double percentageLicenseUsing(Guid License)
    {
        var totalProjectLicense = license.Where(x => x.IdLicense == License).Select(x => x.LicenseType.ProjectAmount)
            .FirstOrDefault();
        var userProjectLicenseUsed = license.Where(x => x.IdLicense == License).Select(x => x.ProjectsLicense.Count
          ()).FirstOrDefault();
        return (Math.Round(((double)userProjectLicenseUsed / (double)totalProjectLicense) * 100, 0));
    }

    public double userAvailable(Guid License)
    {
        var maxUser = license.Where(x => x.IdLicense == License).Select(x => x.LicenseType.UserAmount).FirstOrDefault();
        var currentUser = license.Where(x => x.IdLicense == License).Select(x => x.User).Count();
        return (maxUser - currentUser);
    }

    public double variantsAvailable(Guid License)
    {
        var maxVariants = license.Where(x => x.IdLicense == License).Select(x => x.LicenseType.VariantAmount).FirstOrDefault();
        var currentVariants = license.Where(x => x.IdLicense == License).Select(x => x.ProjectsLicense.Select(x =>
          x.Projects.Apartment.Select(x => x.ApartmentVariants)).Count()).FirstOrDefault();
        return maxVariants - currentVariants;
    }
}
```

Rys.2.7 Zrzut ekranu przedstawiający proces wstrzykiwania zależności. Od teraz każde wywołanie metod obiektu ILicense będzie wywoływało metody w klasie implementującej ten interfejs o nazwie License. Mechanizm wstrzykiwania zależności jest gotowym mechanizmem dostarczonym wraz z ASP.NET Core

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<ILicense, License>();
    services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
        .AddCookie(options =>
        {
            options.LoginPath = "/Home/Login";
            options.Cookie.Name = "Housematica";
        });
    services.AddControllersWithViews();
    services.AddAutoMapper(typeof(Startup));
    services.AddMvc();
    services.AddDbContext<HousematicaContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("HousematicaContext")));
}
```

Rys.2.8. Zrzut ekranu przedstawiający użycie interfejsu ILicense oraz dostarczonych przez niego metod których implementacja znajduje się w klasie License. Wyniki metod przekazywane są do obiektu ViewModel o nazwie LicenseWidgetViewModel oraz zwracane jako model danych widoku częściowego LicenseWidgetComponent.

```
private readonly ILicense _license;

public LicenseWidgetComponent(ILicense license)
{
    _license = license;
}

public async Task<IViewComponentResult> InvokeAsync(Guid License)
{
    var timer = System.Diagnostics.Stopwatch.StartNew();
    LicenseWidgetViewModel licenseWidget = new LicenseWidgetViewModel();
    licenseWidget.UserAvailable = _license.userAvailable(License);
    licenseWidget.PercentageLicenseUsing = _license.percentageLicenseUsing(License);
    licenseWidget.ProjectsAvailable = _license.projectAvailable(License);
    licenseWidget.VariantsAvailable = _license.variantsAvailable(License);
    licenseWidget.LicenseExpired = _license.licenseExpired(License);
    timer.Stop();
    var time = timer.ElapsedMilliseconds;
    return View("LicenseWidgetComponent", licenseWidget);
}
```

Rys.2.9. Zrzut ekranu przedstawiający komponent widoku LicenseWidgetComponent, który zostanie umieszczony w widoku Index i będzie odpowiadał za wyświetlanie okna dialogowego informującego o posiadania licencji w wersji demo.

```
<!-- Modal-->
<button type="button" id="licenseButton" class="btn btn-primary display-none" data-toggle="modal" data-target="#demoLicenceModal">
</button>

<!-- Modal-->
<div class="modal fade" id="demoLicenceModal" tabindex="-1" role="dialog" aria-labelledby="demoLicenceModal" style="display: none;"
  aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered" role="document">
        <div class="modal-content">

            <div class="modal-body text-center align-center align-content-center">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <i aria-hidden="true" class="ki ki-close"></i>
                </button>

                <lottie-player src="/assets/media/lottie/6370-keys.json" background="transparent" speed="1" style="height: 300px;"
                  autoplay></lottie-player>
                <p style="margin-top: 2rem; font-size: 1.1rem;">Wygląda na to, że korzystasz z portalu w wersji demo. Uaktualnij
                  swoją licencję aby uzyskać dostęp do wszystkich funkcji serwisu</p>
            </div>

        </div>
    </div>
</div>
<!-- Modal-->
```

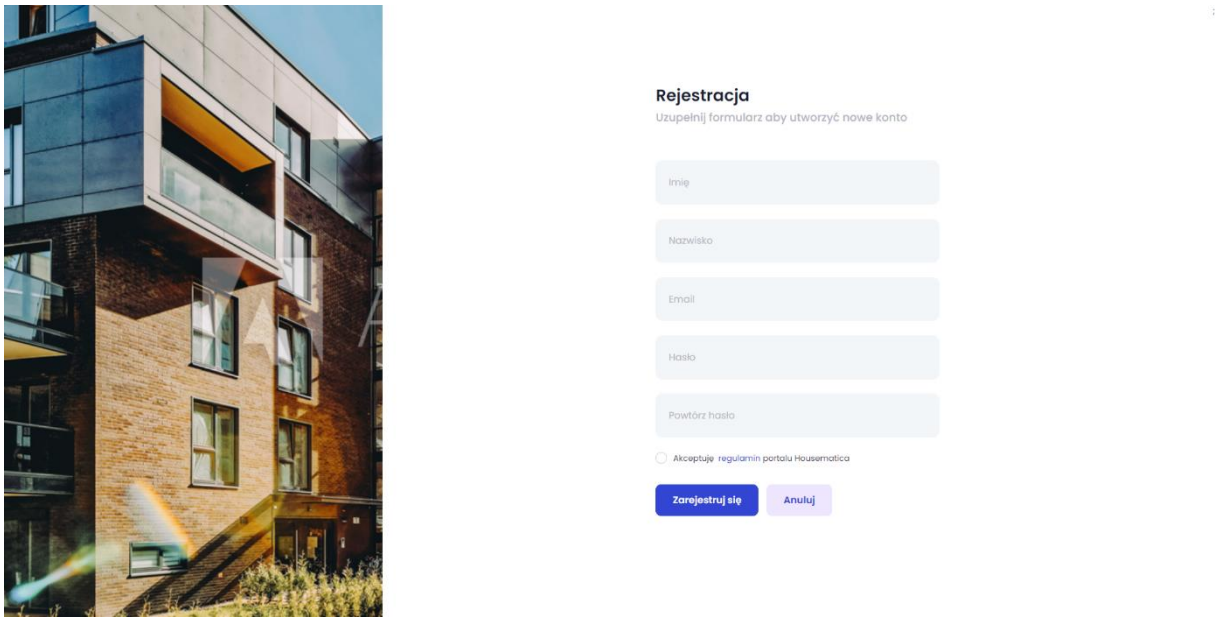
Rys.2.10. Zrzut ekranu przedstawiający przykładowy widok, w tym przypadku widok o nazwie Index. Jak widać na ilustracji, w celu zwiększenia czytelności kodu, zastosowane zostały komponenty, które odpowiadają za różne elementy strony. Ich ładowanie odbywa się asynchronicznie, dzięki czemu zwiększana jest szybkość ładowania się strony.

```
@{
    ViewData["Title"] = "Home Page";
}

@await Component.InvokeAsync("LicenseAlertComponent", ViewBag.User)

<h1 id="json"></h1>
<!--begin::Entry-->
<div class="d-flex flex-column-fluid">
    <!--begin::Container-->
    <div class=" container-fluid ">
        <!--begin::Row-->
        <div class="row">
            <div class="col-xl-3">
                @await Component.InvokeAsync("LicenseWidgetComponent", new Guid(ViewBag.License))
            </div>
            <div class="col-xl-3">
                <!--begin::Widget współpracownicy w projektach-->
                @await Component.InvokeAsync("TeamWidgetComponent", new Guid(ViewBag.License))
                <!--end::Widget współpracownicy w projektach-->
            </div>
            <div class="col-xl-6">
                @await Component.InvokeAsync("NotificationWidgetComponent", new Guid(ViewBag.License))
            </div>
        </div>

        <div class="row">
            <div class="col-xl-6">
                @await Component.InvokeAsync("ProjectsViewWidgetComponent", new Guid(ViewBag.License))
            </div>
            <div class="col-xl-4">
                @await Component.InvokeAsync("ProjectsClickWidgetComponent", new Guid(ViewBag.License))
            </div>
            <div class="col-xl-2">
                @await Component.InvokeAsync("ProjectsFileWidgetComponent", new Guid(ViewBag.License))
            </div>
        </div>
        <!--end::Row-->
    </div>
    <!--end::Container-->
</div>
<!--end::Entry-->

@section Scripts {
    <script>
      var License = "@ViewBag.License";
       $(document).ready(function () {
           if ($("#licenseButton").length) {
               $("#licenseButton").click();
           }
       });

    </script>
    <script src="~/assets/js/pages/dashboardWidgets.js"></script>
    <script src="https://unpkg.com/@{"@"}lottiefiles/lottie-player@latest/dist/lottie-player.js"></script>
}
```

2.4. Schemat działania aplikacji

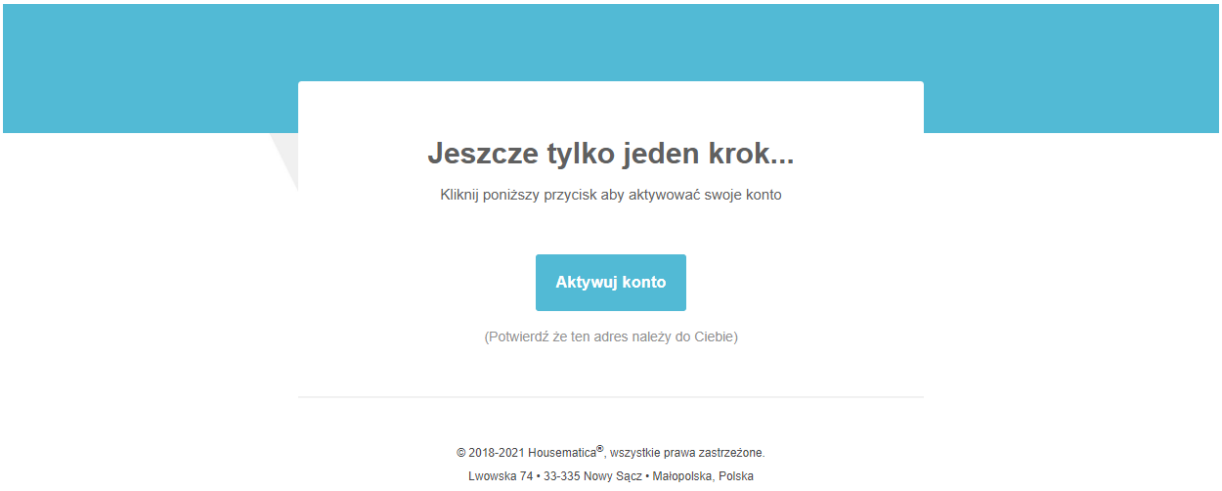
Pierwszym krokiem jest stworzenie przez użytkownika konta w portalu administracyjnym Housematica. W tym celu użytkownik musi podać swoje imię, nazwisko, adres e-mail oraz hasło.

Rys.2.11. Zrzut ekranu przedstawiający formularz rejestracji do portalu Housematica

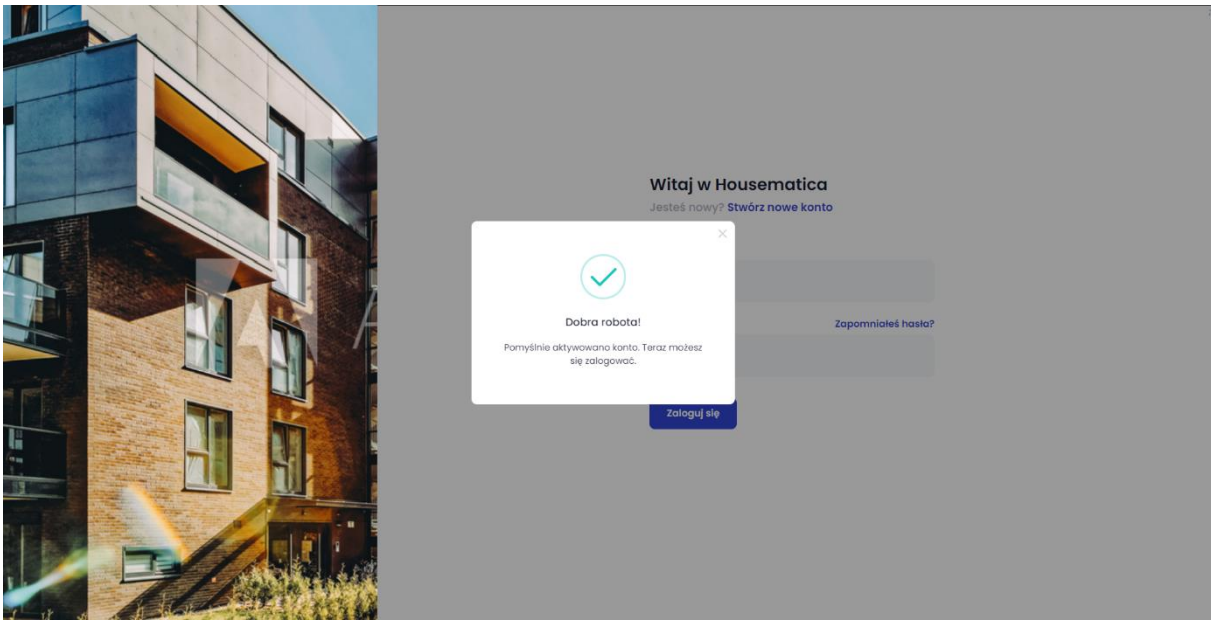


Użytkownik niezwłocznie po utworzeniu konta otrzyma wiadomość e-mail na podany przy rejestracji adres, dzięki której będzie mógł aktywować swoje konto w systemie, otrzymując tym samym dostęp do wszystkich funkcjonalności aplikacji.

Rys.2.12. Zrzut ekranu przedstawiający wiadomość otrzymaną na podany przy rejestracji adres e-mail

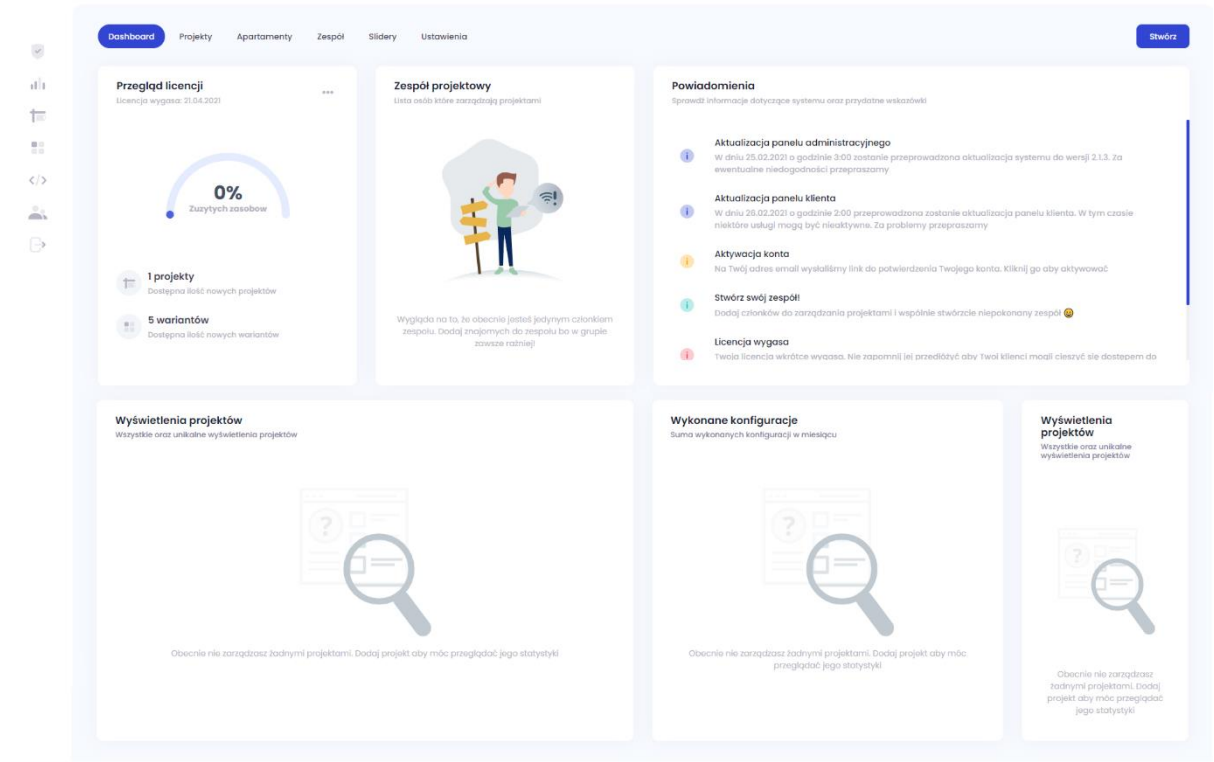


Rys.2.13. Zrzut ekranu przedstawiający monit o aktywacji konta, do którego prowadzi link z wyżej wymienionej wiadomości



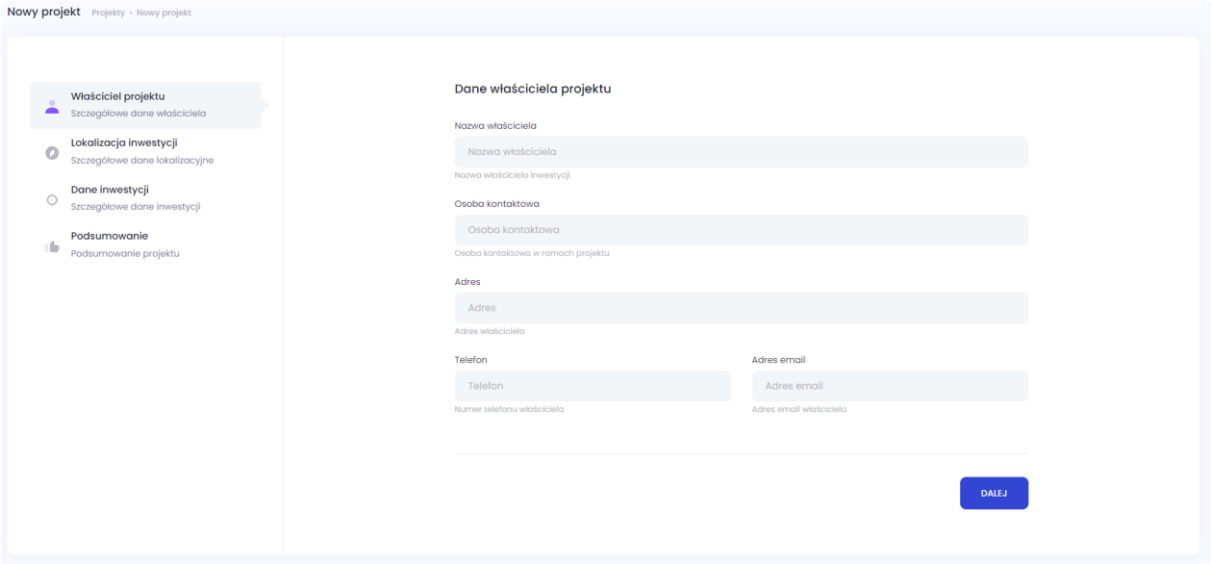
Po aktywacji konta użytkownik może zalogować się na swój profil w panelu administracyjnym.

Rys.2.14. Zrzut ekranu przedstawiający panel klienta po zalogowaniu do portalu administracyjnego

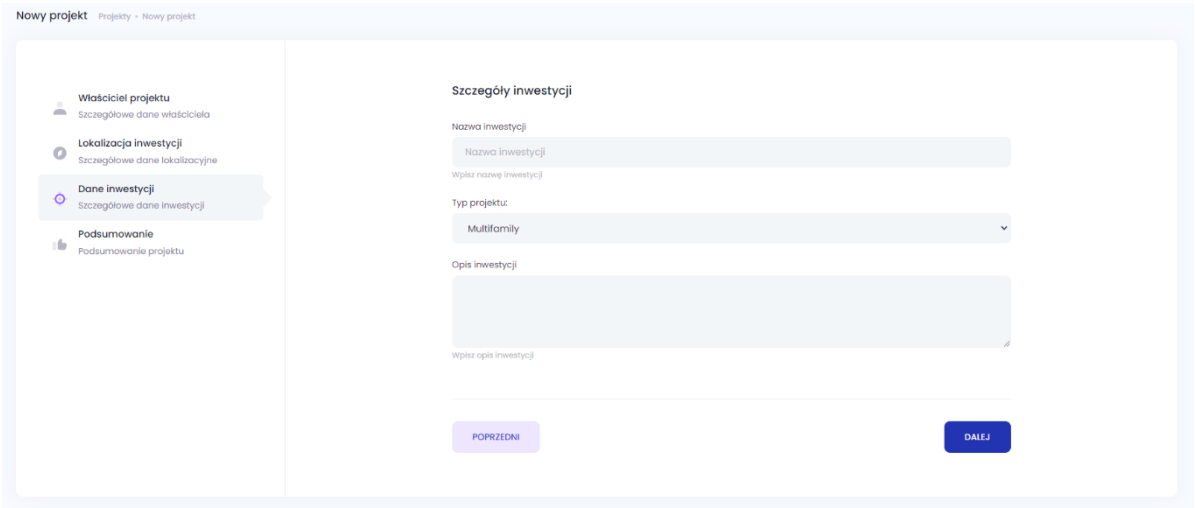


W kolejnym kroku użytkownik może dodać swój pierwszy projekt. Dokona tego klikając w kartę „Projekty”, a następnie „Nowy”.

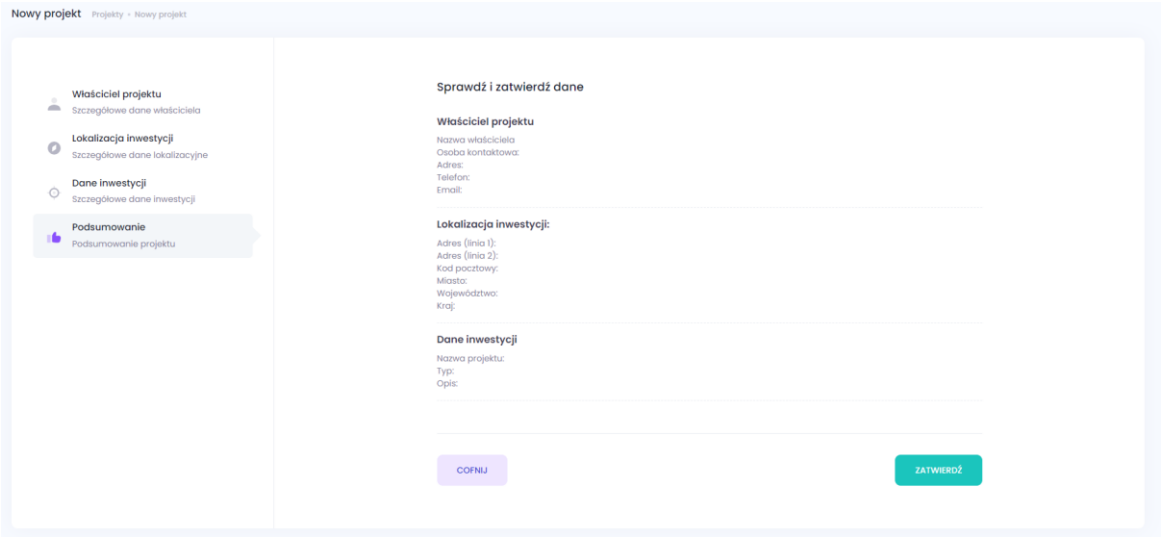
Rys.2.15. Krok 1. – dodanie informacji o właścicielu projektu



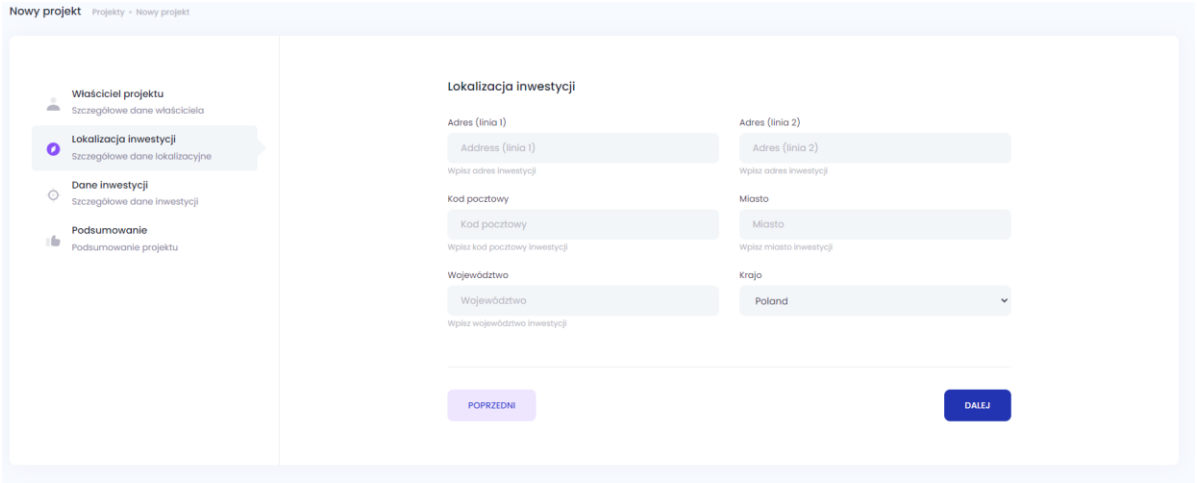
Rys.2.16. Krok 2. – dodanie informacji o lokalizacji inwestycji



Rys.2.17. Krok 3. – dodanie informacji o danych inwestycji



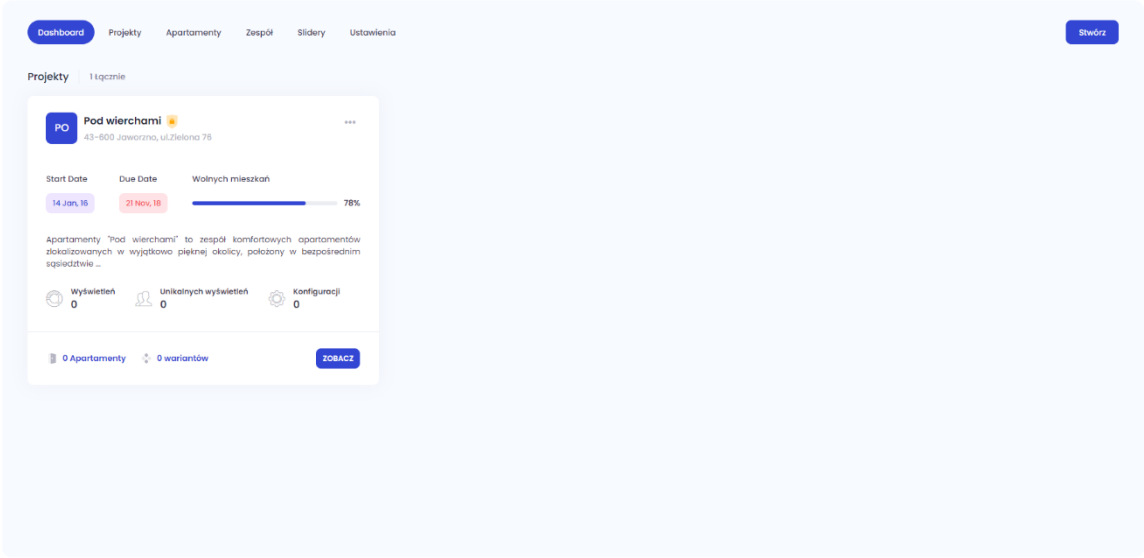
Rys.2.18. Krok 4. – podsumowanie wpisanych informacji



Po dodaniu nowego projektu jego podgląd będzie możliwy w zakładce „Projekty” – „Przeglądaj”. Dodatkowo w tabeli o nazwie Projects w bazie danych wygenerowany zostanie nowy rekord zawierający podane dane projektu oraz automatycznie nadany mu identyfikator w formacie Guid, który pozwoli zdefiniować dany projekt w sposób jednoznaczny. Guid to specjalny format danych, stosowany w przypadkach konieczności nadania obiektowi unikatowego identyfikatora[10].

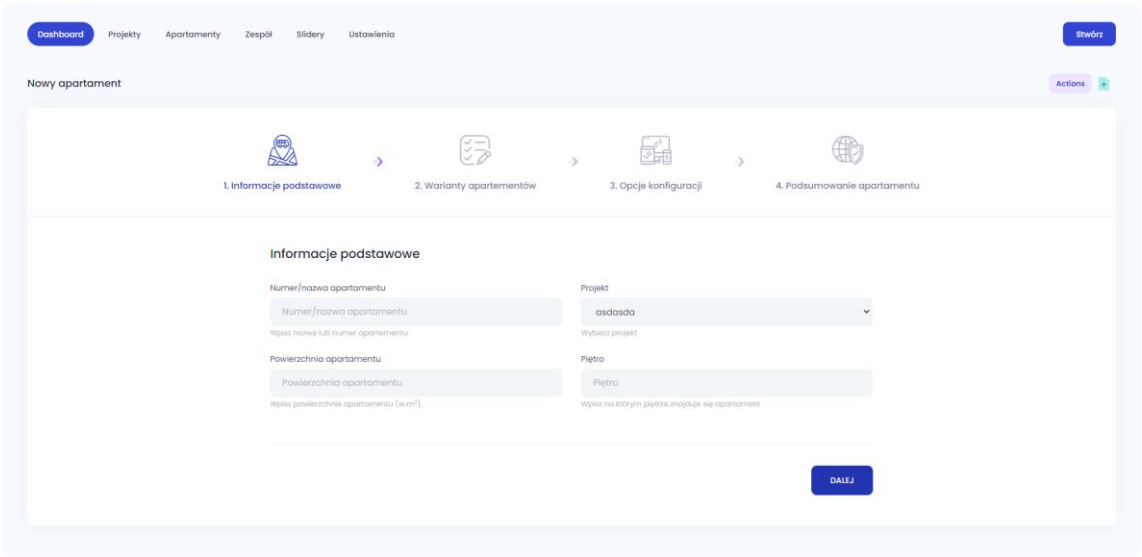
[10] *Guid Struct* Odczytano dnia 24.03.2021, z https://docs.microsoft.com/enus/dotnet/api/system.guid

Rys.2.19. Podgląd projektów użytkownika

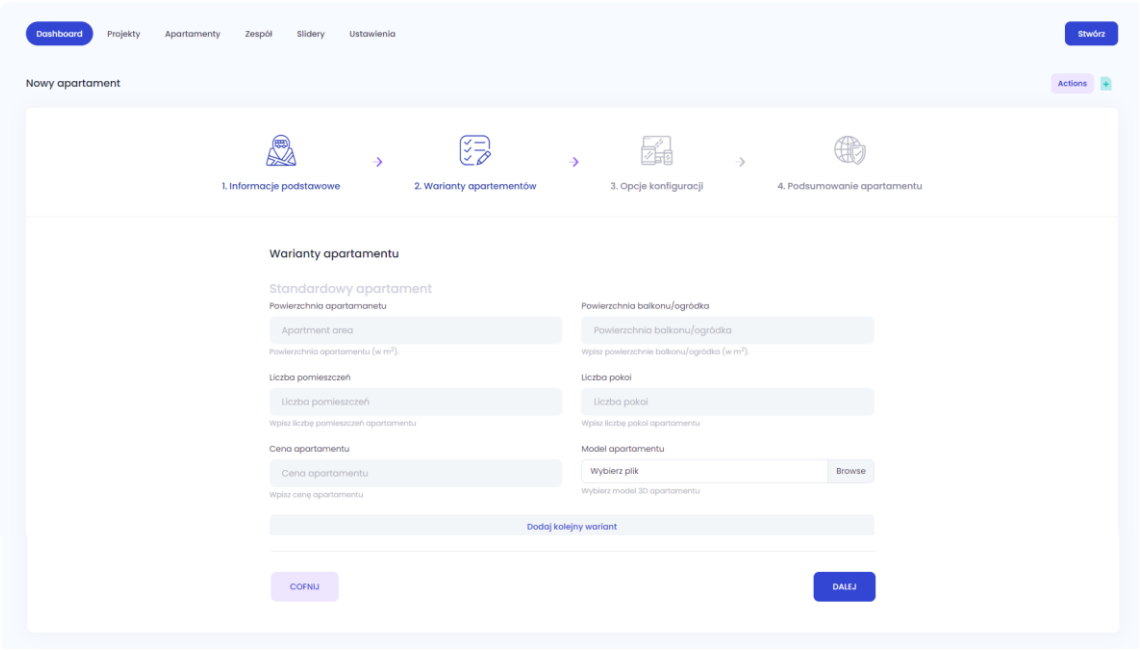


Następnie użytkownik powinien dodać nowy apartament do istniejącego już projektu. Dodawanie nowych apartamentów możliwe jest poprzez przejście do zakładki „Apartamenty” – „Nowy”:

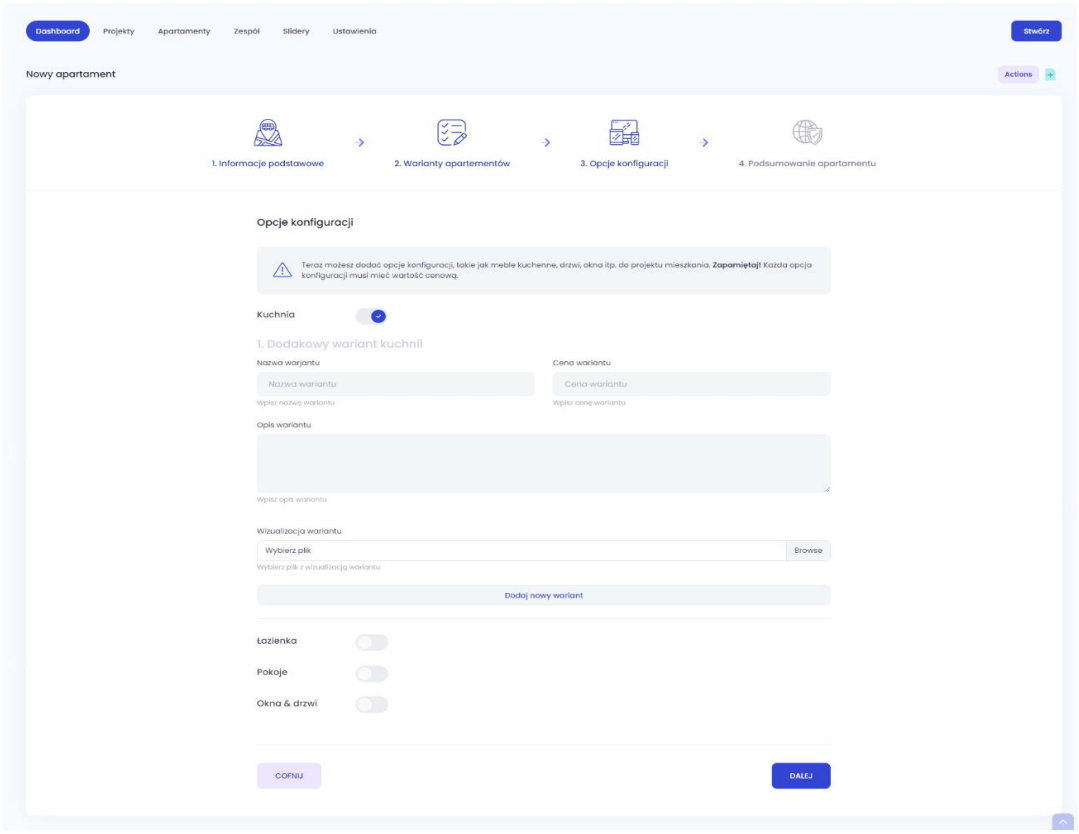
Rys.2.20. Krok 1. – dodanie informacji podstawowych o apartamencie



Rys.2.21. Krok 2. – dodanie standardowej konfiguracji apartamentu oraz opcjonalnie jego dodatkowych wariantów

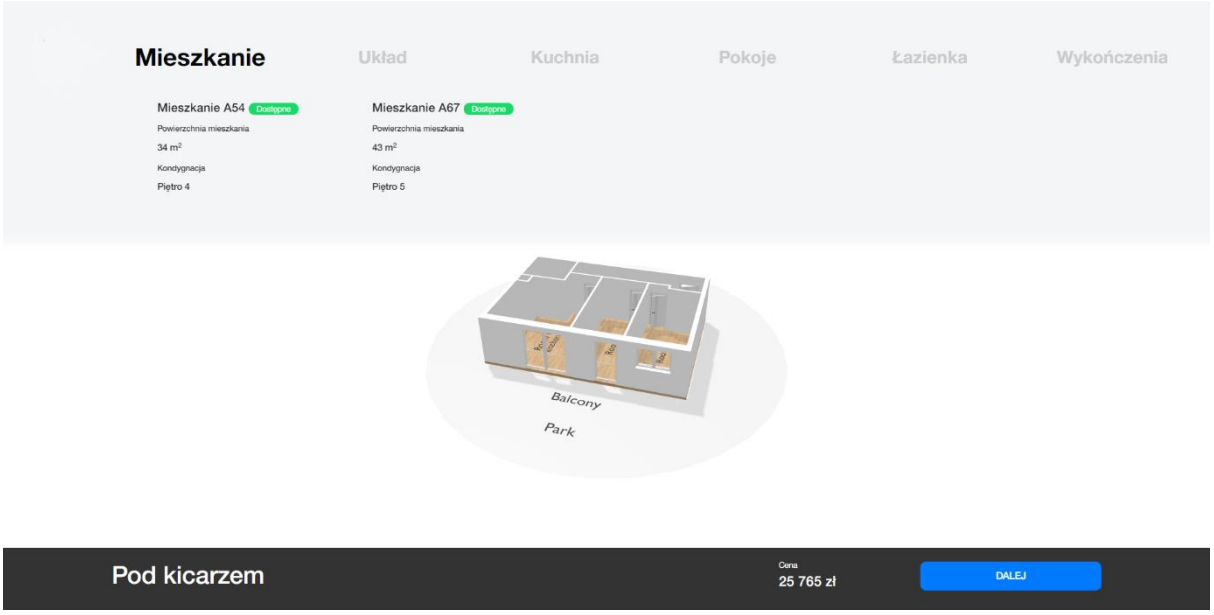


Rys.2.22. Krok 3. – dodanie opcjonalnych konfiguracji kuchni, łazienki, pokojów czy stolarki drzwiowej i okiennej

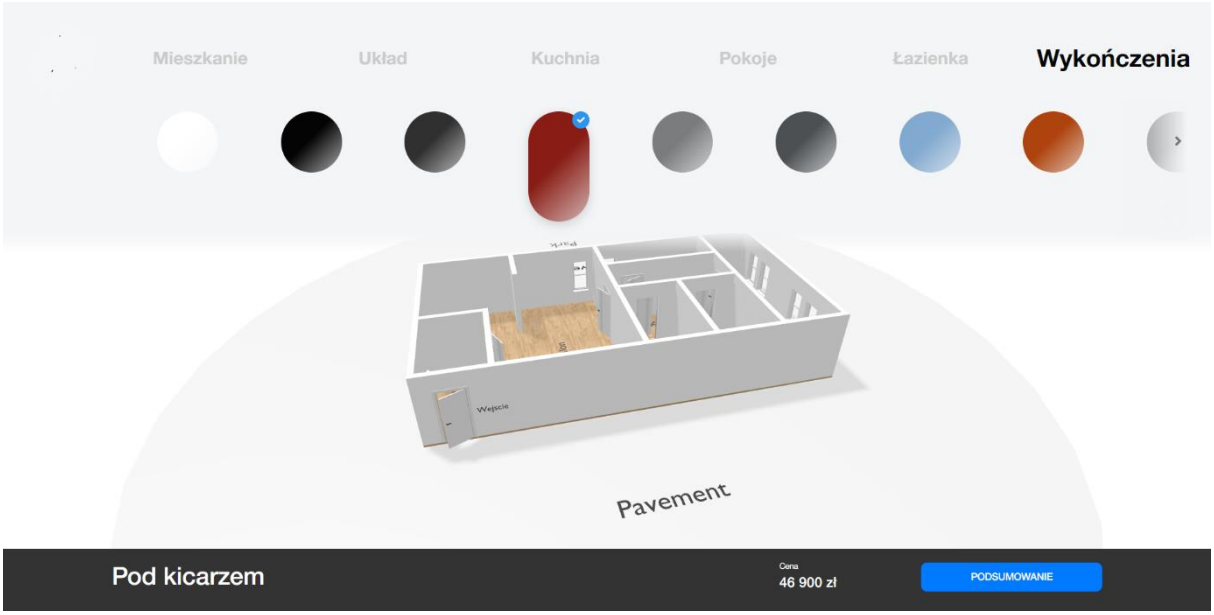


Nowo dodany apartament można zobaczyć w sekcji „Apartamenty” – „Przeglądaj”. Od teraz klienci mogą rozpocząć konfigurowanie apartamentów. W momencie otrzymania adresu przeglądarki do danego projektu, otwarty zostanie panel użytkownika, który umożliwi wybranie apartamentu z puli projektu oraz jego dodatkową konfigurację.

Rys.2.23. Zrzut ekranu przedstawiający panel konfiguracji mieszkania przez klienta



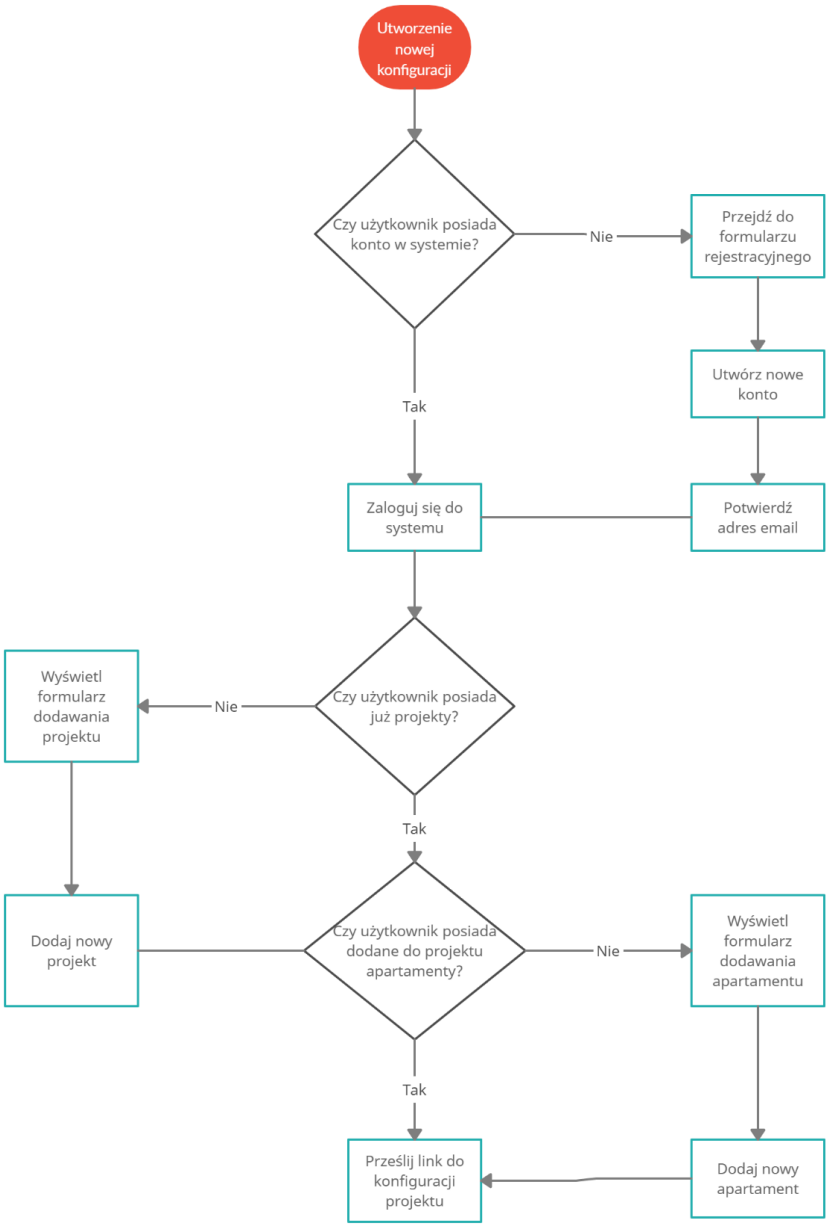
Rys.2.24. Zrzut ekranu przedstawiający wybór jednej z opcji wykończenia elementów mieszkania



2.5. Schemat działania aplikacji – diagram UML

Diagramy UML to schematy przedstawiające zbiór bytów i związków między nimi[11], stanowiące projekt architektury i działania systemów oprogramowania. Za pomocą wizualnych reprezentacji procesu, łatwiej jest zrozumieć możliwe wady lub błędy w oprogramowaniu lub procesach biznesowych. W ramach aplikacji Housematica diagram UML został opracowany na potrzeby zdefiniowania procesu stworzenia nowej konfiguracji, a na jego bazie powstała zakodowana aplikacja Housematica

Rys.2.25. Diagram UML przedstawiający proces utworzenia nowej konfiguracji w ramach projektu



[11] *WIT Wyższa Szkoła Informatyki Stosowanej i Zarządzania Diagramy UML* odczytano dnia 24.03.2021, z http://info.wsisiz.edu.pl/~roksela/materialy/UML/diagramy_uml.pdf

3. Baza danych

Zarówno portal przeznaczony dla klienta, jak i portal do zarządzania projektami, korzysta z jednej bazy danych, składającej się z kilkunastu tabel. W bazie przechowywane są wszystkie dane dotyczące projektów, apartamentów, użytkowników oraz ich licencji i zespołów do jakich przynależą. Baza danych utworzona na potrzeby projektu bazuje na Microsoft SQL Server, czyli systemie do zarządzania relacyjną bazą danych. Jego podstawowym zadaniem jest przechowywanie i udostępnianie danych w ramach żądań danej aplikacji. Aplikacja Housematica korzysta z najnowszej na ten moment wersji systemu SQL Server Microsoft (wersja 15.0.2000.5). Baza danych umieszczona została na zewnętrznym serwerze, dzięki czemu jest ona w efektywny sposób współdzielona pomiędzy portal użytkownika oraz portal administracyjny.

3.1. Założenia przy tworzeniu bazy danych

Głównym założeniem przy projektowaniu bazy danych było stworzenie magazynu danych, który pozwoli w szybki oraz stabilny sposób przetwarzać informacje związane z projektem. Wiele z anomalii, które mogą mieć wpływ na niepoprawne działanie bazy danych rozwiązuje jej normalizacja[12]. Dlatego baza danych użyta w projekcie Housematica została zaprojektowana zgodnie z postulatami Codd'a.

- 1NF (pierwsza postać normalna) – mówi o tym, że każde pole przechowuje wartości atomowe i zawiera tylko jedną wartość (w jednym polu nie możemy umieścić listy wartości). Ponadto, pierwsza postać normalna definiuje posiadanie przez tabelę kluczy głównych – czyli wartości definiujących wiersze w sposób jednoznaczny.

- 2NF (druga postać normalna) – mówi o tym, że każda tabela powinna przechowywać dane dotyczące tylko jednego obiektu. Jeśli tabela posiada kolumny odnoszące się do kilku różnych obiektów, powinna być ona podzielona na kilka mniejszych tabel.

- 3NF (trzecia postać normalna) – mówi o tym, że w tabeli nie należy umieszczać dodatkowych kolumn, których wartości mogą wynikać z operacji na innych kolumnach tej tabeli (np. umieszczanie kolumny kwota vat gdy posiadamy kolumnę wartość i stawka vat)[13].

[12] *Projektowanie i normalizacja bazy* danych odczytano z dnia 23.03.2021, z *https://www.sqlpedia.pl/projektowanie-i-normalizacja-bazy-danych/*
[13] Sharon Allen (2006). *Modelowanie danych* wydawnictwo Helion ISBN: 83-246-0184-8

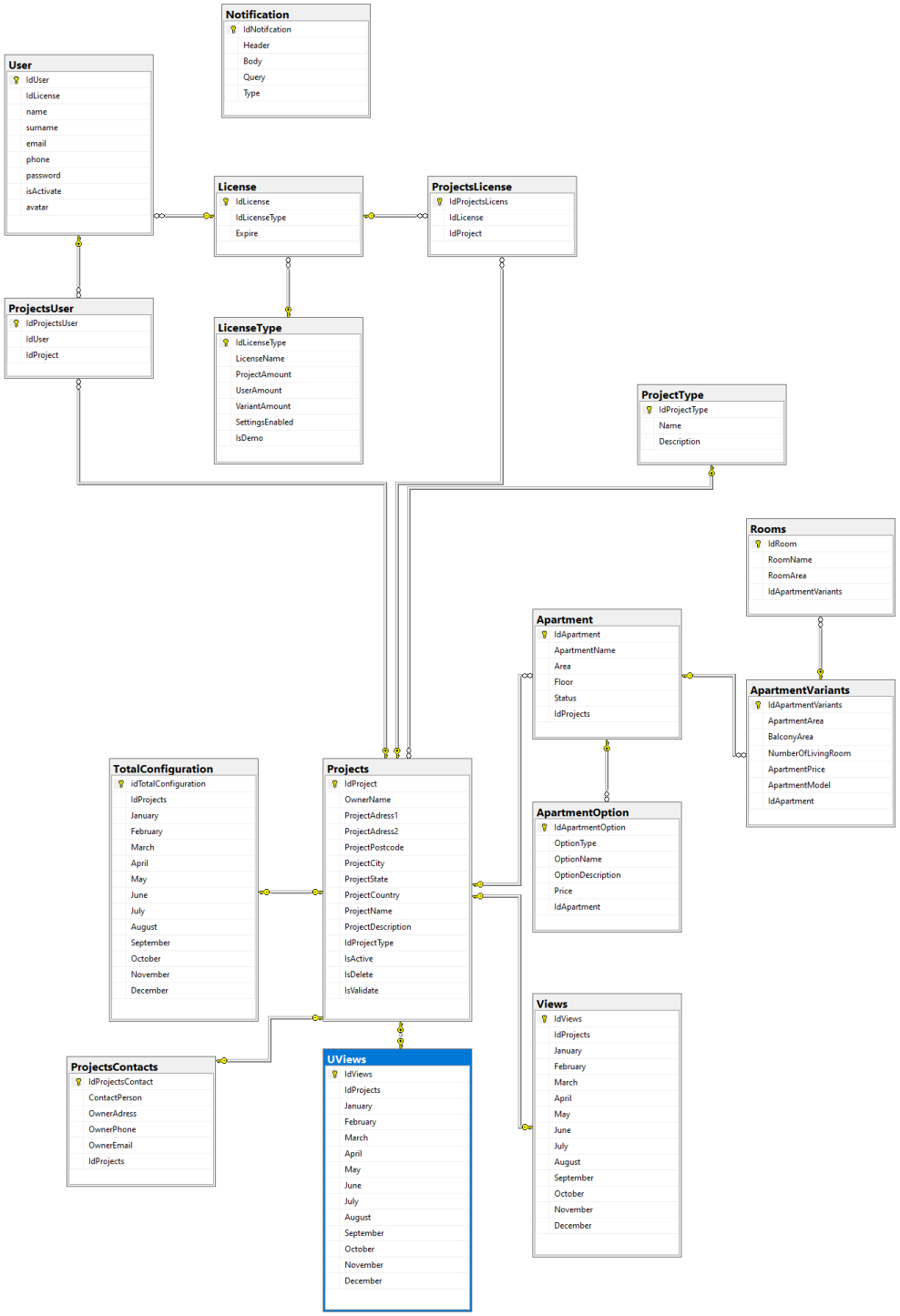
Dzięki zastosowaniu postaci normalnych, baza danych składa się z wąskich, bardziej przejrzystych tabel. Zredukowana została ilość anomalii przy usuwaniu, modyfikowaniu czy dodawaniu danych do bazy. Ulepszony został także proces zarządzania transakcjami. Baza danych używana w aplikacji powinna być dostępna zarówno dla aplikacji użytkownika, jak i administratora, dlatego umieszczona została ona na zewnętrznym serwerze, gdzie oprócz dostępu do niej za pomocą ww. aplikacji, możemy podejrzeć jej strukturę przy użyciu przeglądarki internetowej. Ponadto, dostęp do samej bazy danych powinien zostać specjalnie zabezpieczony, aby uchronić ją przed niepożądanym dostępem osób trzecich. W tym celu użyta została autoryzacja na serwerze, gdzie przechowywana jest baza danych. W celu dodatkowej ochrony bazy danych na serwerze uruchomiony został proces automatycznego tworzenia kopii bazy danych, która wysyłana jest na serwer bazujący na innej maszynie. Dodatkowo kopia struktury bazy danych oraz polecenia potrzebne do jej utworzenia przechowywane są w aplikacji, dzięki czemu baza może być w szybki sposób odtworzona oraz, w razie potrzeby, w łatwy sposób edytowana.

Rys.3.1. Zrzut ekranu przedstawiający przykładową migrację w projekcie Housematica. W tym przypadku wygenerowana automatycznie migracja tabeli o nazwie ProjectsLicense. Dzięki wykonaniu migracji, struktura bazy danych zostanie zaktualizowana o nową tabelę o nazwie ProjectsLicense.

```
migrationBuilder.CreateTable(
    name: "ProjectsLicense",
    columns: table => new
    {
        IdProjectsLicens = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        IdLicense = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
        IdProject = table.Column<Guid>(type: "uniqueidentifier", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_ProjectsLicense", x => x.IdProjectsLicens);
        table.ForeignKey(
            name: "FK_ProjectsLicense_License_IdLicense",
            column: x => x.IdLicense,
            principalTable: "License",
            principalColumn: "IdLicense",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_ProjectsLicense_Projects_IdProject",
            column: x => x.IdProject,
            principalTable: "Projects",
            principalColumn: "IdProject",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateIndex(
    name: "IX_ProjectsLicense_IdLicense",
    table: "ProjectsLicense",
    column: "IdLicense");

migrationBuilder.CreateIndex(
    name: "IX_ProjectsLicense_IdProject",
    table: "ProjectsLicense",
    column: "IdProject");
```

3.2. Projekt bazy danych

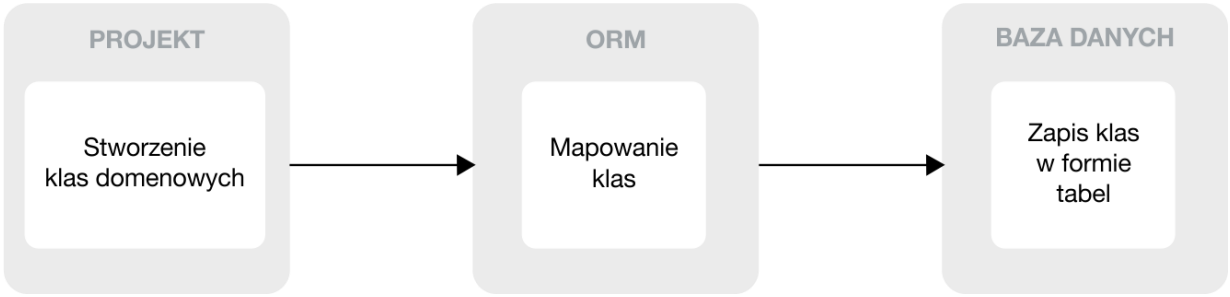
Rys.3.2. Diagram bazy danych wygenerowany za pomocą Microsoft SQL Management Studio



3.3. Code first

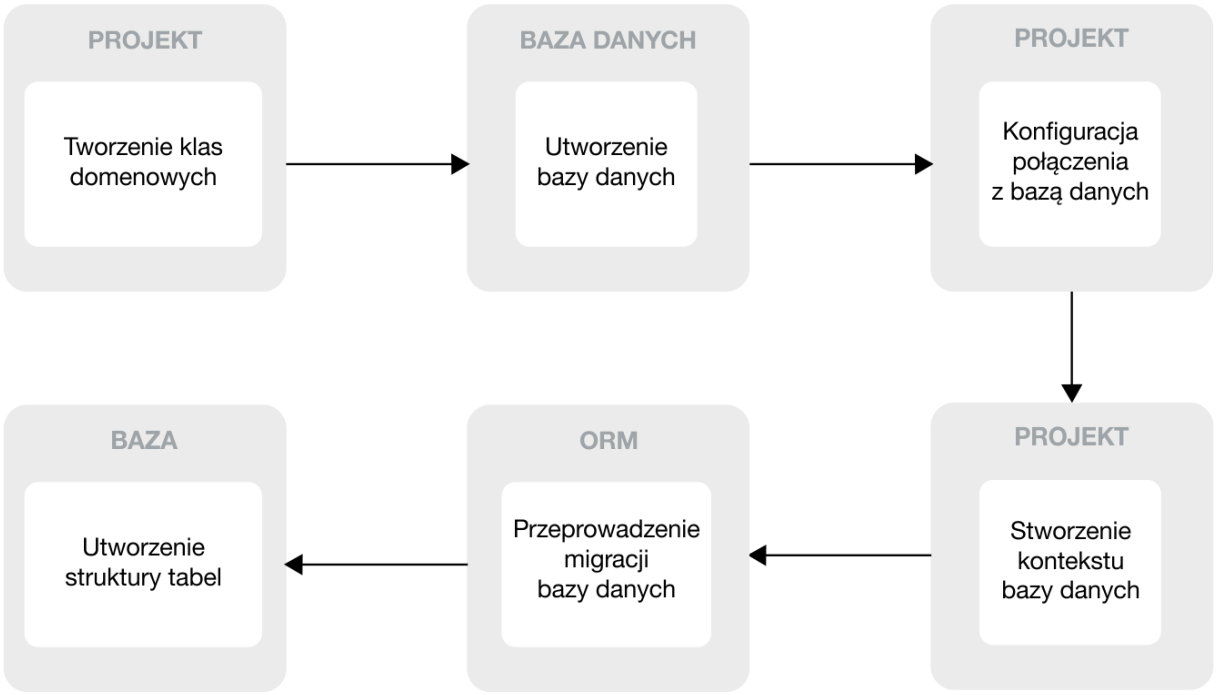
Na potrzeby projektu Housematica baza danych została stworzona przy pomocy koncepcji *Code first*. Jej założenia polegają na wcześniejszym stworzeniu klas domenowych, które za pomocą oprogramowania typu ORM zostaną odwzorowane w bazie danych jako obiekty tabeli. Jako oprogramowanie typu ORM użyty został Entity Framework w wersji 4.1. Pozwala on na skoncentrowaniu się na domenie aplikacji i stworzeniu klasy dla jednostki domeny, zamiast najpierw projektować bazę danych, a następnie tworzyć klasy, które pasują do projektu bazy danych.

Rys.3.3. Schemat tworzenia obiektów bazy danych w koncepcji Code First



Za pomocą Entity Framework została także zrealizowana obsługa danych, taka jak: modyfikacja, usuwanie, dodawanie oraz przetwarzania rekordów tabeli.

Rys.3.4. Workflow tworzenia tabel przy użyciu Entity Framework w przypadku projektu Housematica



4. Projekt aplikacji do konfigurowania budynków

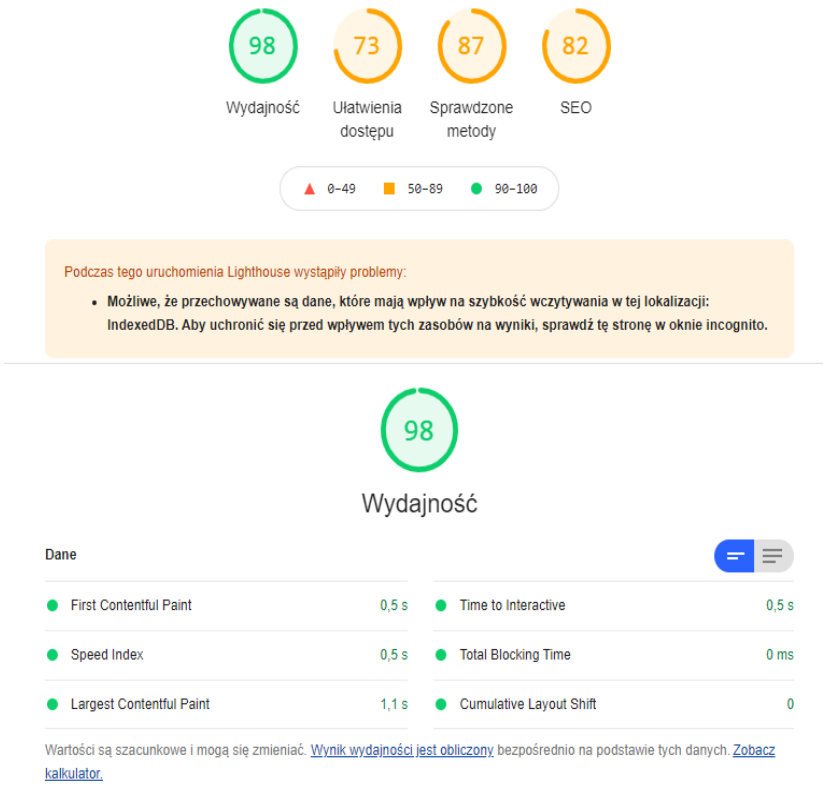
Jedną z integralnych części projektu jest stworzenie aplikacji webowej, umożliwiającej użytkownikowi za pomocą każdej przeglądarki obsługującej HTML5 konfigurowanie apartamentów za pomocą wcześniej przygotowanych modeli 3D. Ponadto, użytkownik powinien otrzymać informację o szacunkowych kosztach konfiguracji oraz wybranych przez niego modyfikacjach. Jedną z najważniejszych funkcji konfiguratora powinna być możliwość przeglądania modelu apartamentu z różnych kątów oraz poziomu zbliżenia/oddalenia.

4.1. Założenia przy tworzeniu aplikacji do konfigurowania budynków oraz ich implementacja w projekcie

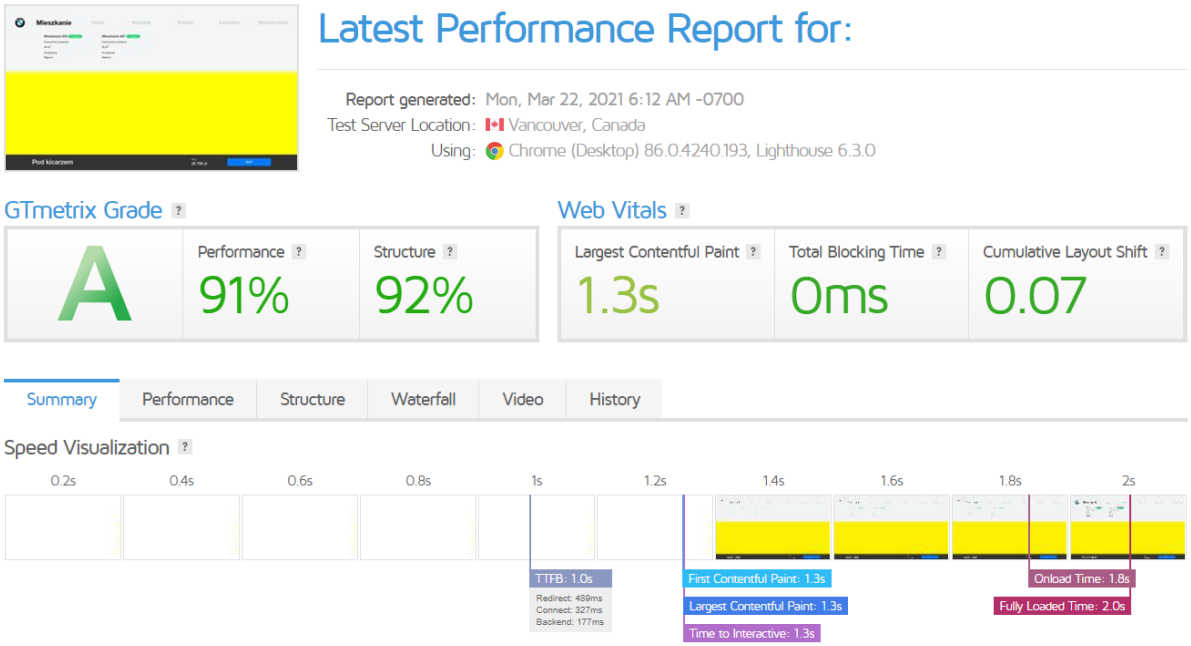
Kluczowym czynnikiem przy tworzeniu aplikacji jest czas jej ładowania. Według badań przeprowadzonych przez Google[14] czas ładowania strony jest jednym z głównych czynników który decyduje o pozostaniu użytkownika na stronie lub jego opuszczeniu jej. Z racji, iż w aplikacji ładowane będą modele 3D (które z założenia cechują się dużą wielkością), sama struktura aplikacji została maksymalnie odchudzona do użycia tylko wymaganych oraz niezbędnych do funkcjonowania przez aplikację bibliotek. Do projektu dołączona została zminifikowana biblioteka Bootstrap, a niektóre elementy (takie jak alerty przeglądarki) zostały przepisane na nowo, bez użycia dodatkowych bibliotek. Ponadto wszystkie czcionki znajdujące się w aplikacji webowej zostały przekonwertowane do lekkiego formatu WOFF2. WOFF2 to format czcionki, który zapewnia redukcję rozmiaru pliku średnio o 30%[15], pomagając w ten sposób szybciej ładować czcionki internetowe w zgodnych przeglądarkach.

[14] *Tools for Web Developers – Lighthouse* odczytano 25 marca 2020 z developers.google.com/web/tools/lighthouse
[15] *What is WOFF2 and how do I enable it for my projects?* odczytano 25 marca 2020 z *https://www.fonts.com/support/faq/what-is-woff2-and-how-do-i-enable-it*

Rys.4.1. Zrzut ekranu przedstawiający szybkość ładowania aplikacji zbadany za pomocą narzędzia Lighthouse Google

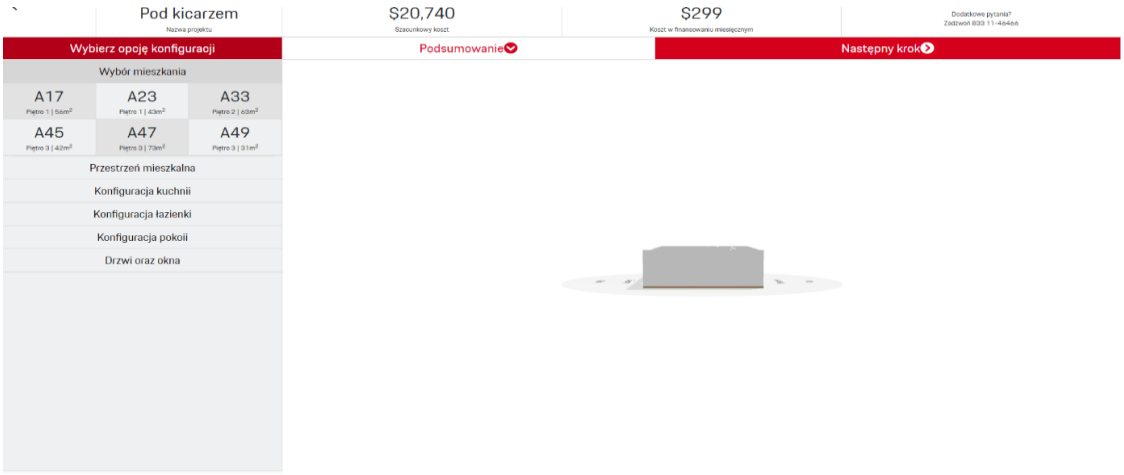


Rys.4.2. Zrzut ekranu przedstawiający szybkość ładowania aplikacji zbadany za pomocą narzędzia GTmetrix

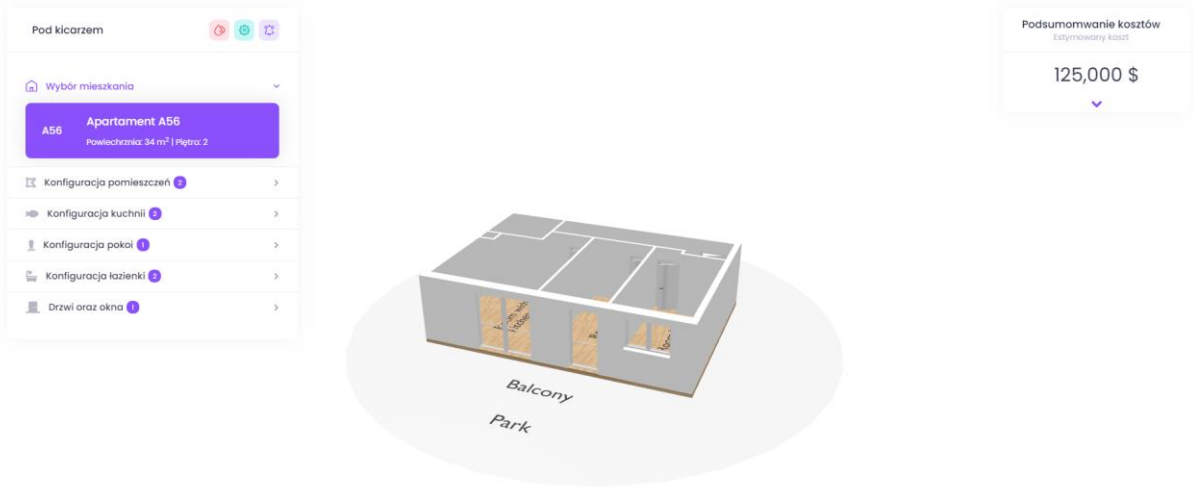


Kolejnym czynnikiem, decydującym o pozytywnym postrzeganiu aplikacji webowej przez odbiorców, jest jego wygląd oraz intuicyjność działania. Portal klienta Housematica został zaprojektowany zgodnie z panującymi trendami odnośnie projektowania interfejsu użytkownika. Użyte czcionki są czytelne oraz odpowiednio kontrastujące z tłem aplikacji webowej. Przełączaniu opcji towarzyszą animacje, dzięki którym korzystanie z interfejsu wydaje się bardziej płynne. Ponadto, niezależnie od skalowania okna przeglądarki, interfejs użytkownika zachowuje swoje poprawne proporcje. Finalna wersja interfejsu użytkownika aplikacji wielokrotnie ewaluowała, aby ostatecznie stworzyć jak najbardziej przejrzysty i intuicyjny interfejs.

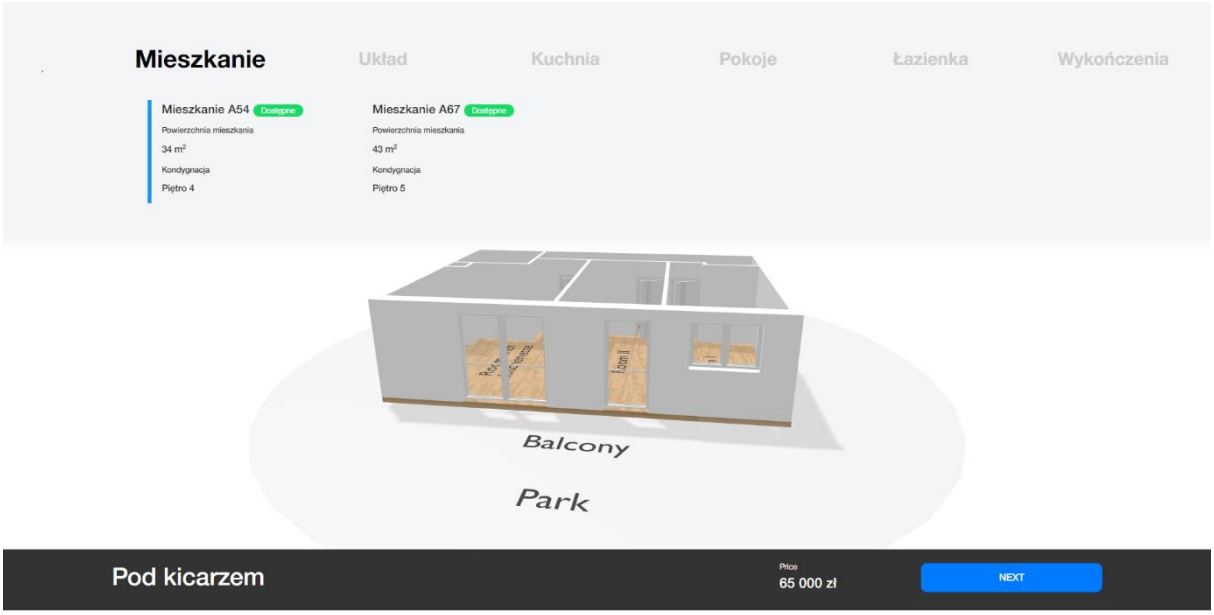
Rys.4.3. Housematica panel klienta – wersja I



Rys.4.4. Housematica panel klienta – wersja II



Rys.4.5. Zrzut ekranu przedstawiający interfejs użytkownika aplikacji webowej Housematica – część przeznaczona dla klienta – wersja III



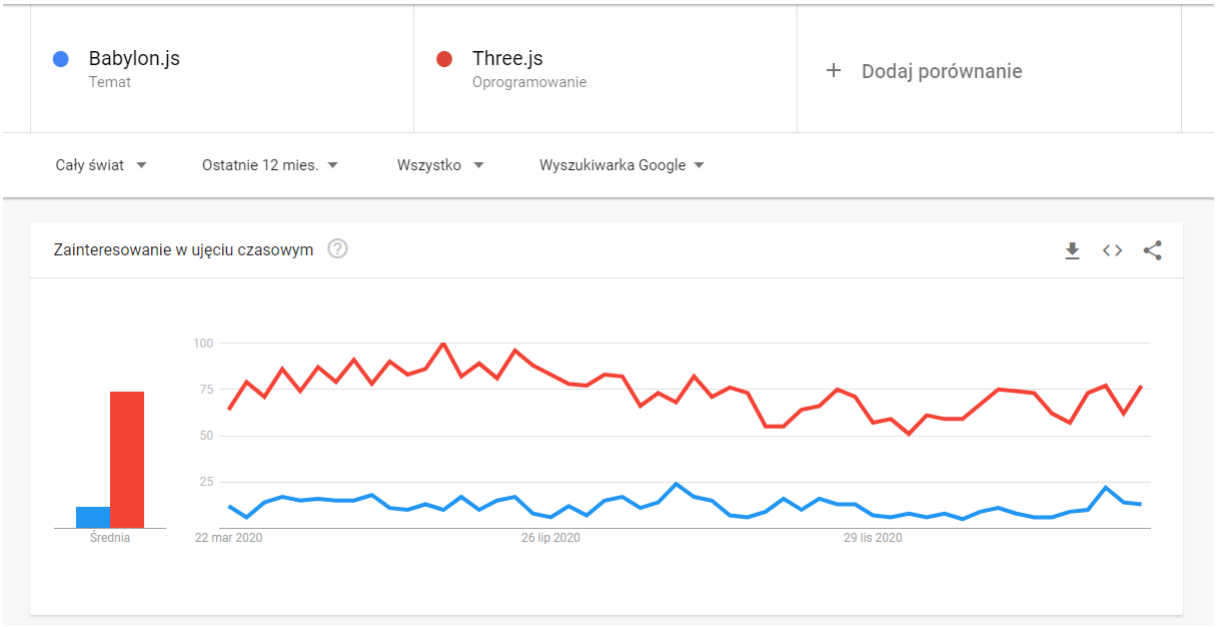
4.2.Obsługa modeli 3D

Aplikacja Housematica korzysta z wcześniej przygotowanych przez administratora projektu modeli 3D. Na potrzeby projektu modele wykonane zostały w oprogramowaniu do tworzenia grafiki 3D o nazwie Blender, oraz wyeksportowane do formatu .babylon za pomocą narzędzia BlenderExporter (narzędzie dostępne pod adresem repozytorium GitHub: https://github.com/BabylonJS/BlenderExporter). Pomimo użycia w projekcie modeli z rozszerzeniem .babylon, wykorzystana technologia do wyświetlania modeli 3D pozwala także na używanie szerokiego spektrum rozszerzeń plików 3D, takich jak: glTF czy OBJ.

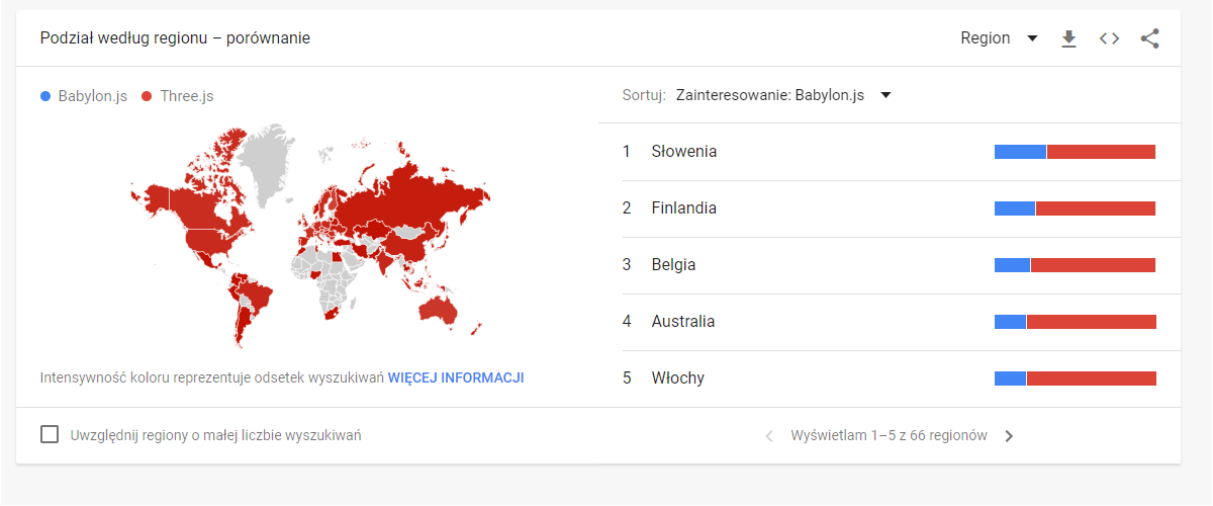
Do wyświetlania modeli 3D został użyty silnik renderujący Babylon.js. Pomimo, że nie jest on najpopularniejszym silnikiem do tego typu operacji, to zrzesza on wokół siebie szerokie grono społeczności bardzo aktywnej na forum tej technologii. Zastosowany proces modelowania 3D polega na modelowaniu wielokątów z trójkątnymi bokami, które mają być reprezentowane przez modele powłokowe. Po utworzeniu modele są renderowane na elemencie canvas HTML 5 za pomocą programu cieniującego, który określa pozycje pikseli i kolory tekstur zastosowanych do każdego modelu, kamery sceny i świateł wraz z matrycami światowymi 4 x 4 dla każdego obiektu, który przechowuje ich położenie i skalę[16].

[16] *Building Shaders With Babylon.js* odczytano dnia 25.03.2021 z https://www.smashingmagazine.com/2016/11/building-shaders-with-babylon-js/

Rys.4.6 Porównanie najpopularniejszego silnika renderującego pliki 3D z silnikiem Babylon.js – screen I

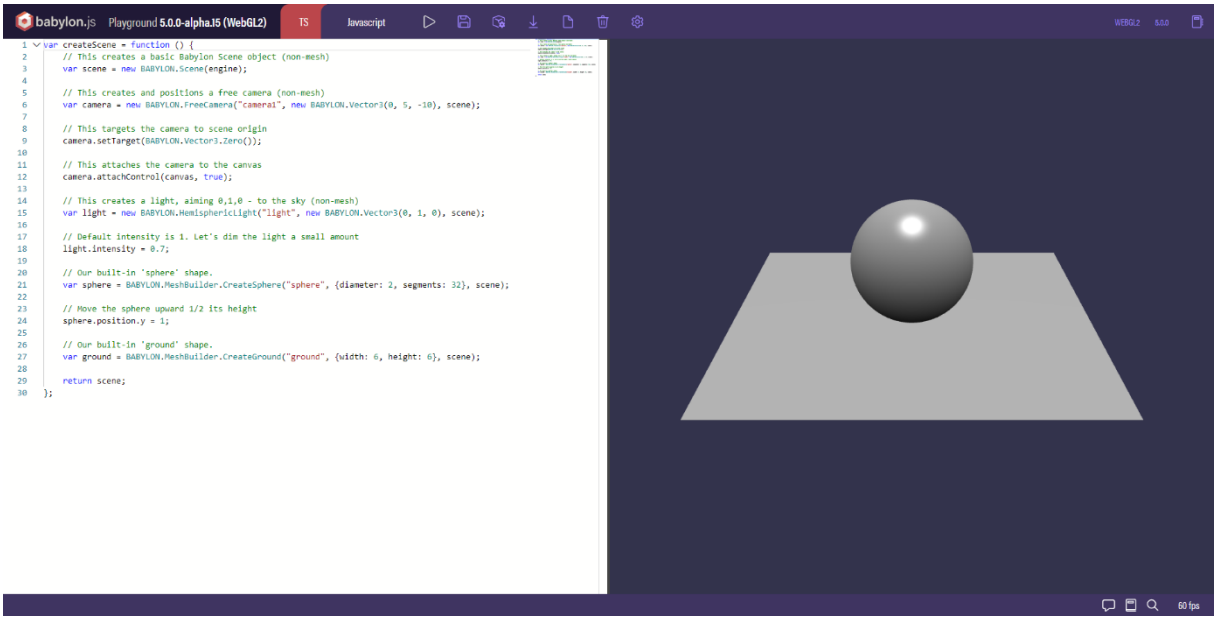


Rys.4.7. Porównanie najpopularniejszego silnika renderującego pliki 3D z silnikiem Babylon.js – screen II



Sama praca z silnikiem Babylon.js jest stosunkowo prosta, a dzięki dostarczonej przez twórców silnika „piaskownicy", w przeglądarce można w łatwy i szybki sposób testować kod aplikacji Babylon.js. Niewątpliwie zaletą tego silnika jest także rozbudowana, lecz przejrzysta dokumentacja, która rozwiązuje większość typowych problemów z obsługą modeli 3D.

Rys.4.8. Webowa wersja „piaskownicy" przygotowanej przez twórców Babylon.js



Zasada użycia silnika Babylon.js w projekcie Housematica jest następująca: na początku stworzony zostaje obiekt typu Canvas, do którego zaczytywane będą modele 3D apartamentów. Następnie, do każdej pobranej bazy danych przypisywany jest atrybut opisujący daną opcję w sposób jednoznaczny. Po wybraniu danej opcji uruchamiany jest skrypt, który przeładowuje model apartamentu w zależności od wybranej opcji konfiguracji. Ostatecznie, do sekcji Canvas ładowany zostaje odpowiedni model 3D bez przeładowania interfejsu użytkownika.

Rys.4.9. Obiekt Canvas w którym uruchomiony zostanie model 3D. Domyślnie silnik Babylon.js renderuje obiekty 3D wyszukując elementu o identyfikatorze „renderCanvas". Element w którym renderowany jest model można w łatwy sposób podmienić w skrypcie renderującym obiekty.

```
<!--begin: Main View-->
<div class="main-view"><canvas id="renderCanvas"></canvas></div>
<!--end: Main View-->
```

Rys.4.10. Inicjalizacja silnika renderującego, sceny oraz elementu Canvas. Obiekt 3D zostanie wyrenderowany w elemencie o identyfikatorze „renderCanvas".

```
var canvas = document.getElementById("renderCanvas");

var engine = null;
var scene = null;

var sceneToRender = null;
var createDefaultEngine = function() { return new BABYLON.Engine(canvas, true,
    { preserveDrawingBuffer: true, stencil: true }); };
```

Rys.4.11. Funkcja delayCreateScene() odpowiada za stworzenie nowej sceny oraz dodania jej do tabeli przechowywującej wszystkie sceny. Dodatkowo na końcu metody tworzona jest nowa kamera, ustawiana jej pozycja, a ponadto określana czułość przybliżania obrazu.

```
function delayCreateScene() {
var scene1 = new BABYLON.Scene(engine);
scenes = [];
scenes.push(scene1);
var currentScene = scene1;
var camera1 = new BABYLON.ArcRotateCamera("camera1",  0, 0, 0, new BABYLON.Vector3(0, 0, -0), scene1);
camera1.wheelprecision = 24000;
```

Rys.4.12. Inicjalizacja modelu 3D (osadzony statycznie), światła oraz warstw. Dodatkowo za pomocą metod scene1.activeCamera ustawiany jest maksymalny kąt obrotu kamery. Metoda BABYLON.ShadowGenerator odpowiada za wygenerowanie cieni obiektów znajdujących się na płaszczyźnie.

```
scene1.createDefaultCameraOrLight(true, true, true);
scene1.clearColor = new BABYLON.Color3(255,255,255);
scene1.activeCamera.lowerBetaLimit = 1;
scene1.activeCamera.upperBetaLimit = 1.5;
var light = new BABYLON.DirectionalLight("light1", new BABYLON.Vector3(-2, -6, -2), scene1);
    light.position = new BABYLON.Vector3(20, 60, 20);
    light.shadowMinZ = 30;
    light.shadowMaxZ = 180;
    light.intensity = 1;
    var generator = new BABYLON.ShadowGenerator(512, light);
    generator.useContactHardeningShadow = true;
    generator.bias = 0.01;
    generator.normalBias= 0.05;
    generator.contactHardeningLightSizeUVRatio = 0.08;
    for (var i = 0; i < scene1.meshes.length; i++) {
        generator.addShadowCaster(scene1.meshes[i]);
        scene1.meshes[i].receiveShadows = true;
        if (scene1.meshes[i].material && scene1.meshes[i].material.bumpTexture) {
            scene1.meshes[i].material.bumpTexture.level = 2;
        }
    }
camera1 = scene1.activeCamera;
});
```

Rys.4.13. Na ilustracji blok kodu odpowiadający za próbę uruchomienia silnika renderującego. Metoda uruchamiająca silnik jest gotową metodą dostarczoną w dokumentacji silnika Babylon.js.

```
var engine;
try {
engine = createDefaultEngine();
} catch(e) {
console.log("the available createEngine function failed. Creating the default engine instead");
engine = createDefaultEngine();
}
if (!engine) throw 'engine should not be null.';
scene = delayCreateScene();;
sceneToRender = scene

engine.runRenderLoop(function () {
    if (sceneToRender) {
        sceneToRender.render();
    }
});

// Resize
window.addEventListener("resize", function () {
    engine.resize();
});
```

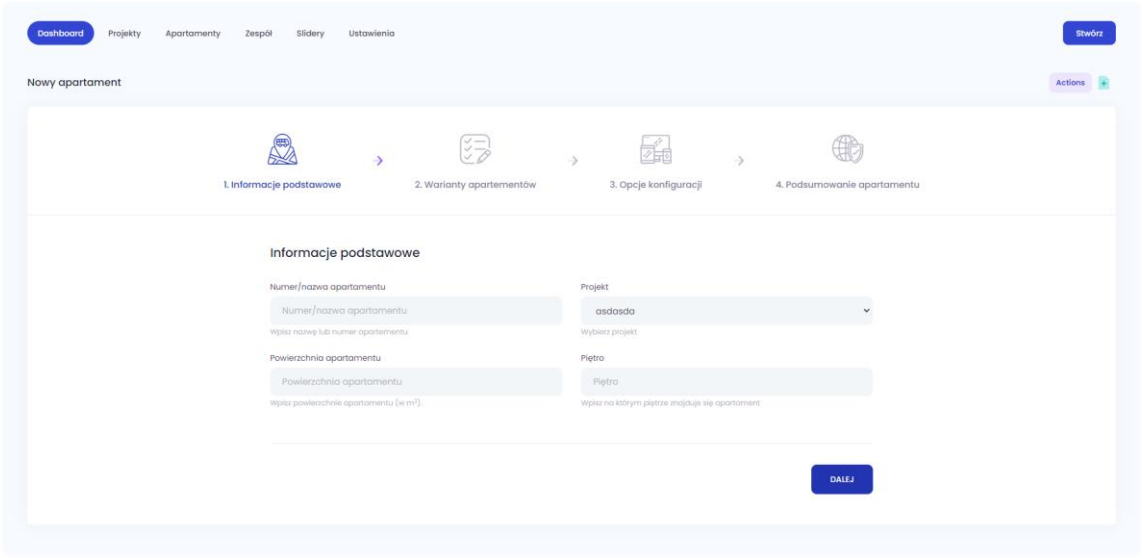
5. Panel administracyjny

Drugim filarem projektu Housematica jest stworzenie panelu administracyjnego do zarządzania projektami oraz ich łatwej edycji czy usuwania. Na rynku nie istnieje rozwiązanie, które pozwalałoby klientowi bez udziału firmy programistycznej oraz specjalistycznej wiedzy dotyczącej programowania stworzyć konfigurator mieszkania. Ponadto panel administracyjny powinien być wyposażony w podstawowe funkcjonalności, dotyczące zarządzania projektem oraz przeglądaniem jego statystyk i informacji z nim związanych.

5.1. Założenia przy tworzeniu panelu administracyjnego aplikacji do konfigurowania budynków oraz ich implementacja w projekcie

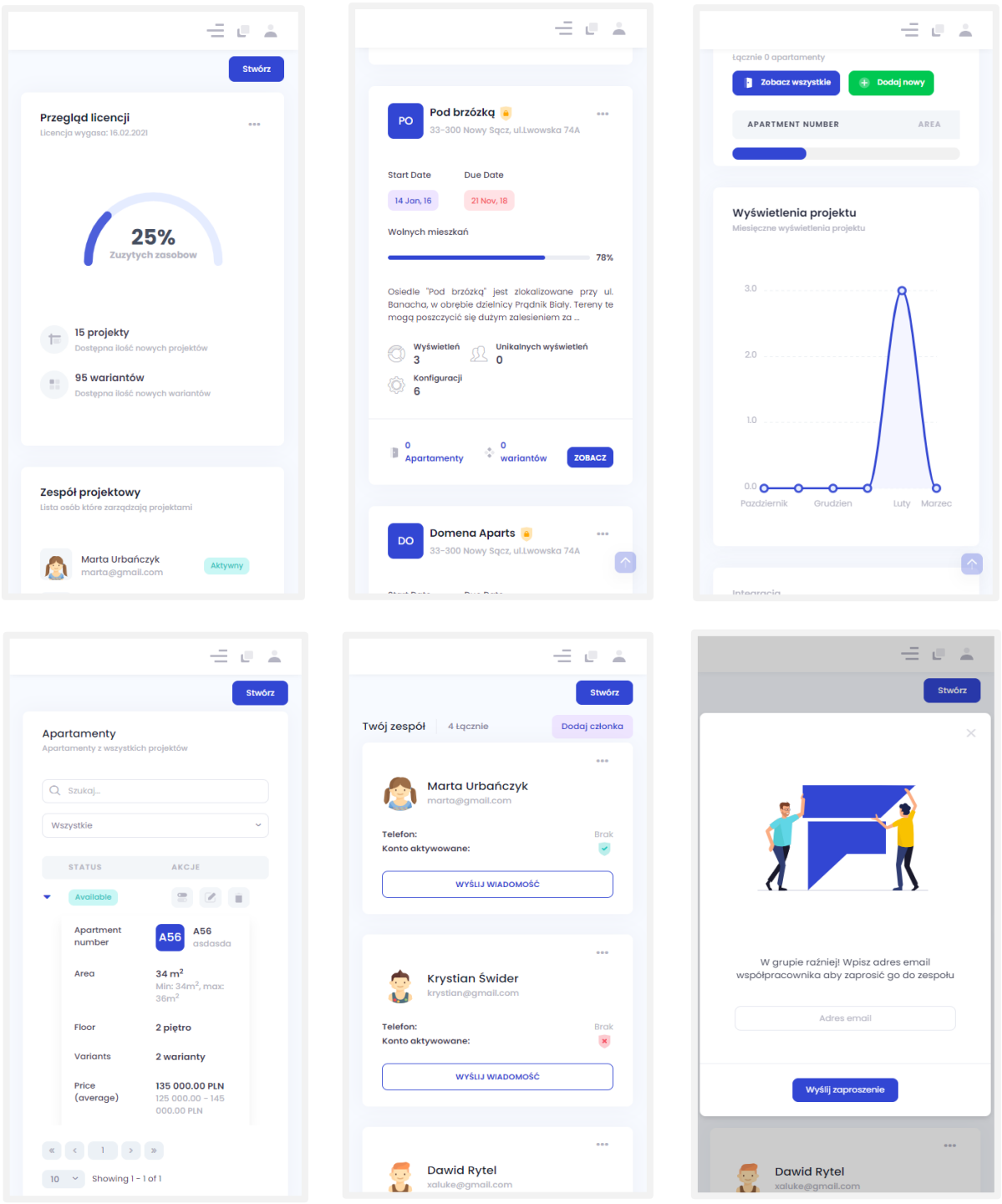
Pierwszym, kluczowym celem panelu administracyjnego, który wykreuje przewagę pomiędzy projektem Housematica a innymi tego typu rozwiązaniami na rynku, było stworzenie przystępnego panelu do dodawania nowych projektów oraz, w ich obrębie, apartamentów i ich opcji. W panelu administracyjnym Housematica zarówno dodawanie projektów, jak i apartamentów odbywa się za pomocą czytelnego formularza, który przeprowadzając użytkownika przez kolejne kroki, pozwala na stworzenie gotowego konfiguratora mieszkania. Klient, oprócz posiadania modelu 3D mieszkania, potrzebuje jedynie podstawowej wiedzy na temat inwestycji, takiej jak: jej nazwa, opis czy lokalizacja. Reszta zadań odbywa się już za pomocą zaprogramowanej logiki biznesowej.

Rys.5.1 Przykładowy widok przedstawiający prostotę formularza dodawania nowego apartamentu



Ponadto, drugim niezwykle ważnym wymogiem panelu administracyjnego, było dostosowanie go do urządzeń mobilnych. Przy pomocy urządzeń mobilnych, administratorzy projektu powinni mieć dostęp do wszystkich kluczowych funkcji programu, a szczególną wagę należy przywiązać do przeglądania statystyk projektu, zarządzania członkami jego zespołu a także listą dostępnych projektów.

Rys.5.2. Zrzuty ekranu przedstawiające wygląd aplikacji na urządzeniach mobilnych

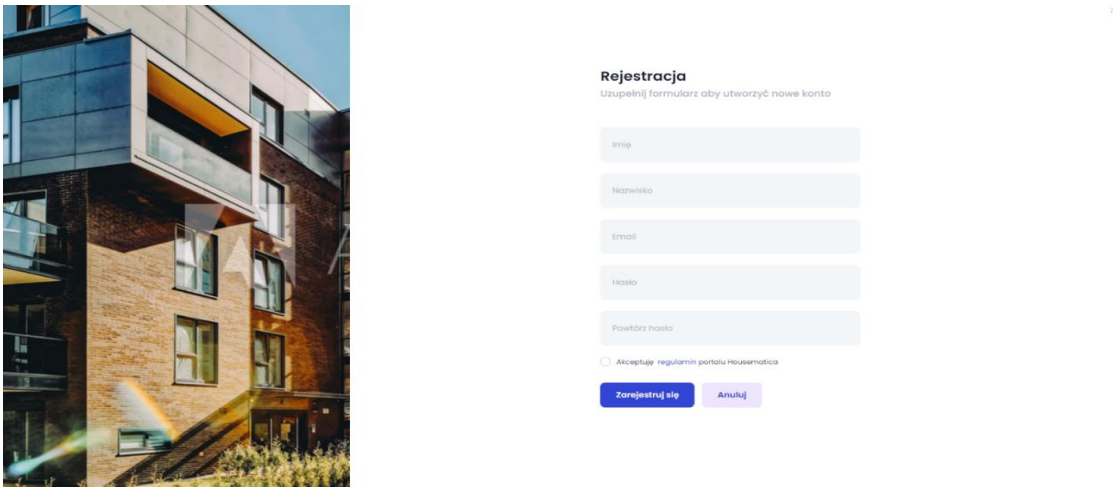


5.2. Dodatkowe funkcjonalności panelu administracyjnego

Na elementy panelu administracyjnego składać się będą:

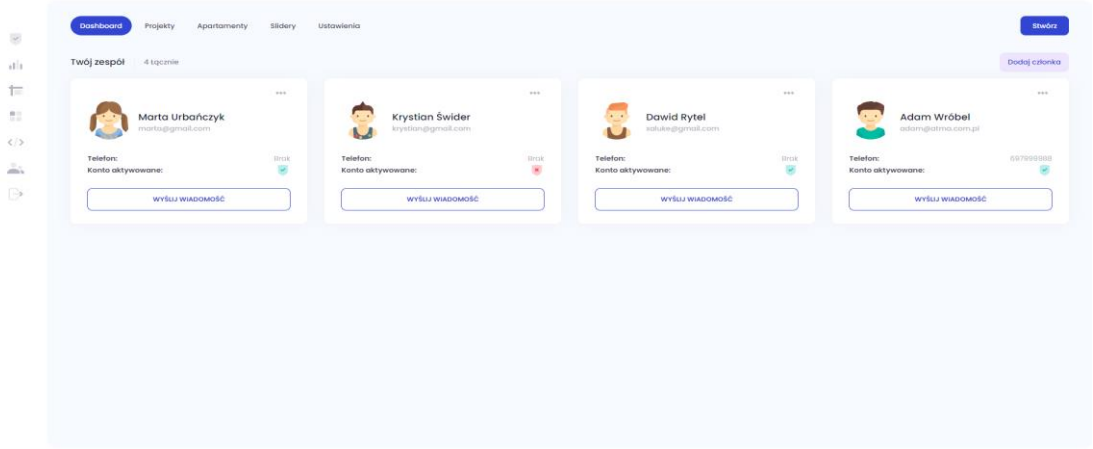
— Autoryzacja – klienci mogą rejestrować oraz logować się do systemu Housematica. Proces autoryzacji odbywa się za pomocą mechanizmu ASP.NET Core Identity. Przy rejestracji nowego konta, użytkownik otrzymuje wiadomość e-mail pozwalającą aktywować jego konto.

Rys.5.3. Zrzut ekranu przedstawiający panel Rejestracji do systemu



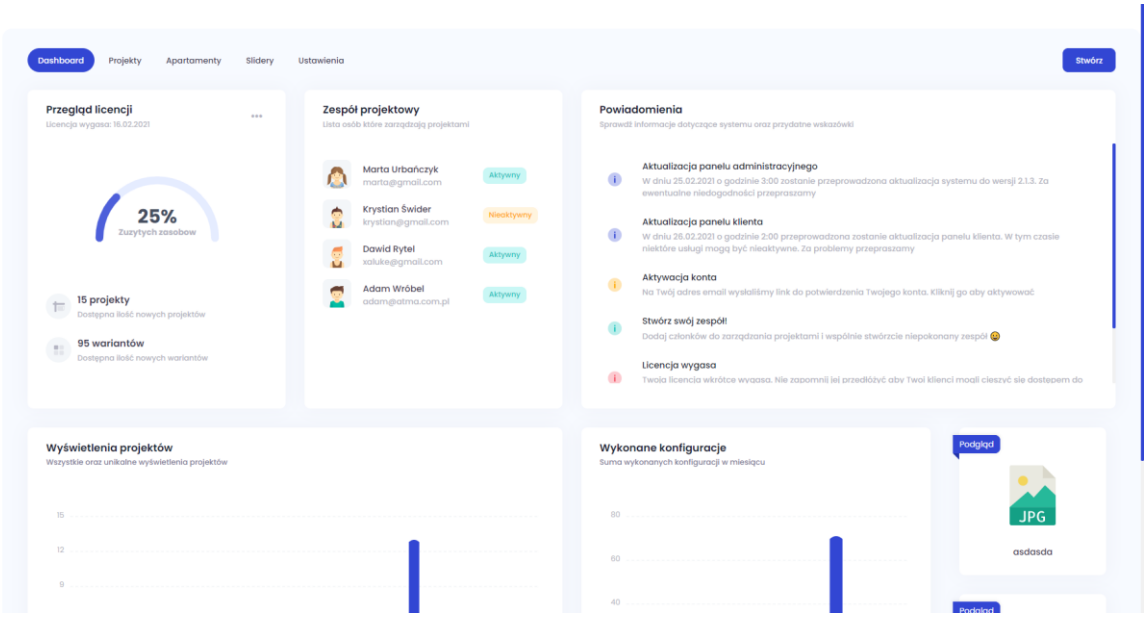
— Zespoły – klienci mogą tworzyć zespoły poprzez zapraszanie osób do skorzystania z platformy. Proces zapraszania odbywa się poprzez wpisanie adresu e-mail przyszłego członka zespołu. Na podany adres e-mail zaproszony otrzyma powiadomienie o możliwości założenia konta i dołączenia do zespołu.

Rys.5.4. Zrzut ekranu przedstawiający panel zespołu



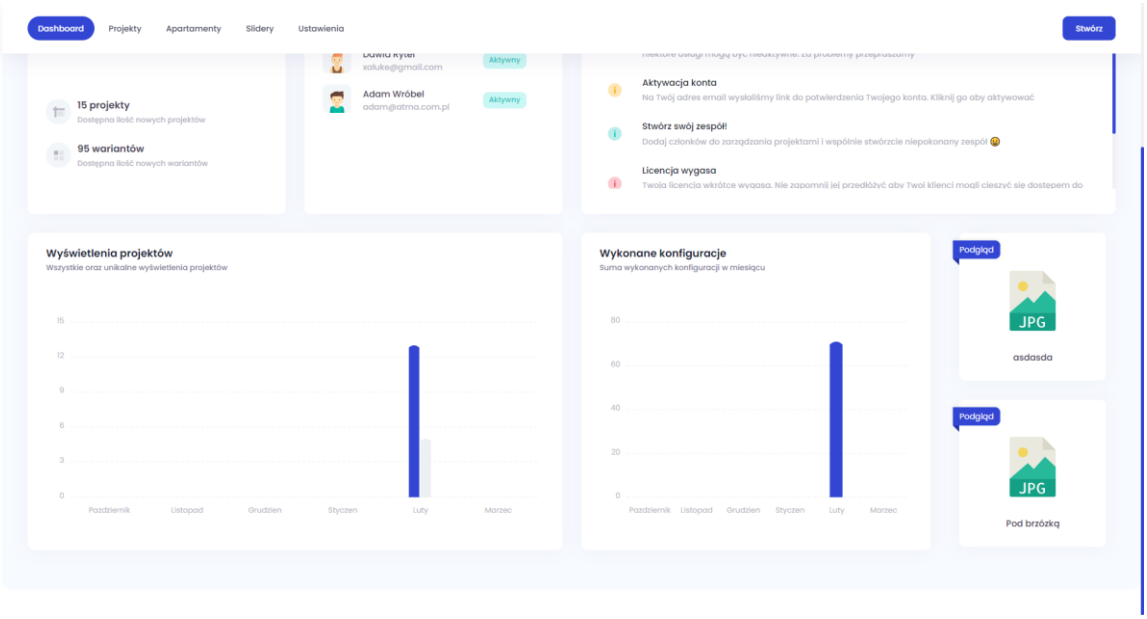
— Licencje – portal administracyjny działa na zasadzie licencji. Każdy zespół posiada własną licencję, która pozwala na dodawanie określonej liczby nowych projektów, apartamentów czy członków zespołu.

Rys.5.5. Zrzut ekranu przedstawiający widget wykorzystanych oraz pozostałych operacji w ramach licencji



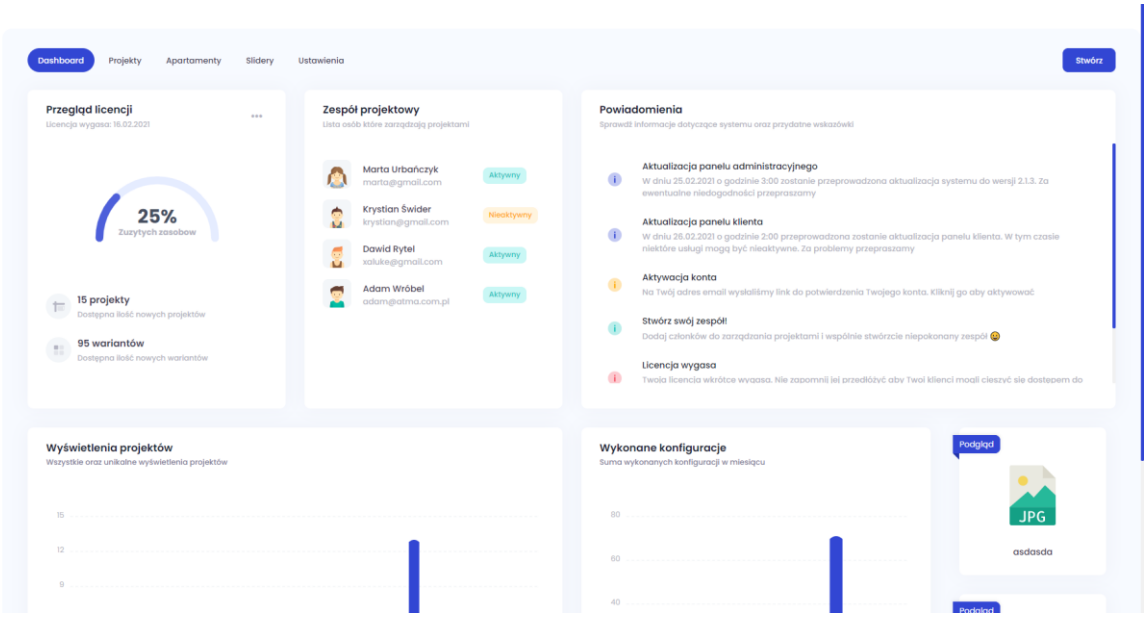
— Statystyki – klienci mogą przeglądać ilość wyświetleń projektów, unikalnych wyświetleń oraz ilość wykonanych konfiguracji ich projektów.

Rys.5.6. Zrzut ekranu przedstawiający widget statystyk projektów użytkownika



— Powiadomienia – panel administracyjny portalu Housematica został wyposażony w moduł powiadomień, który pozwala na przesyłanie alertów do użytkowników o przyszłych aktualizacjach systemu czy wygasających licencjach.

Rys.5.7. Zrzut ekranu przedstawiający widget powiadomień



6. Testy aplikacji

Testowanie aplikacji to kluczowy etap w procesie tworzenia aplikacji biznesowej. W trakcie poddania aplikacji testom, możemy wykryć wiele błędów w oprogramowaniu oraz w jego logice działania, zanim trafi on do środowiska produkcyjnego. Testy dzielimy na pięć poziomów[17]:

- testy modułowe (jednostkowe),
- testy integracyjne wewnętrzne,
- testy systemowe,
- testy integracyjne zewnętrzne,
- testy akceptacyjne (odbiorcze).

Rys.6.1. Piramida poziomów testów. Kolejność testów powinna być wykonywana z oddolną interpretacją infografiki. Opracowanie własne bazujące na źródłach [18] [19]

6.1. Testy modułowe w projekcie Housematica

Testowanie modułowe definiuje typ testowania oprogramowania, w którym sprawdzana jest poprawność działania wyizolowanych fragmentów kodu [20]. Testy modułowe polegają na

[17] Rafał Pawlak (2014). *Testowanie oprogramowania* wydawnictwo HELION, ISBN: 978-83-246-9308-5
[18] Vladimir Khorikov (2020). *Testy jednostkowe. Zasady, praktyki i wzorce* wydawnictwo HELION, ISBN: 978-83-283-6871-2
[19] Rafał Pawlak (2014). *Testowanie oprogramowania* wydawnictwo HELION, ISBN: 978-83-246-9308-5
[20] *What is Module Testing? Definition, Examples* Odczytano dnia 25.03.2021, z https://www.guru99.com/module-testing.html

wywołaniu danego fragmentu kodu, w celu weryfikacji czy wykonuje on poprawnie przydzielone mu operacje. Tego rodzaju testy zwykle wykonuje się w środowisku developerskim, gdzie programista posiada dostęp do napisanego kodu. Testowanie modułowe jest niezwykle istotnym procesem w trakcie tworzenia oprogramowania, ponieważ błędy wykryte na początku etapu wytwarzania oraz wdrażania oprogramowania, wymagają o wiele mniej czasu na ich poprawę, aniżeli poprawa nieprawidłowości, gdy aplikacja wdrożona jest już do środowiska produkcyjnego. Im wcześniej program z mniejszą ilością błędów zostanie przeniesiony do fazy formalnych testów, tym szybciej sam projekt trafi do końcowego etapu oraz wzrośnie jego jakość[21]. Nowoczesnym podejściem do tworzenia testów modułowych jest koncepcja „najpierw testuj". Jest to podejście, które opiera się na pojedynczych cyklach tworzenia przypadków testowych, następnie budowaniu i integracji niewielkich fragmentów kodu, wykonywaniu testów modułowych, poprawianiu usterek i powtarzaniu tego procesu aż testy zostaną zaliczone [22].

W ramach projektu Housematica przeprowadzone zostały modułowe testy najważniejszych oraz najbardziej narażonych na awarię części systemu. Testy projektu Housematica bazują na frameworku NUnit, który dostarcza platformę testową dla całego środowiska .NET, a ponadto pozwala na równoległe testowanie wielu fragmentów kodu[23].

Rys.6.2. Przykładowy test sprawdzający poprawność zwracania odpowiedniego widoku przez kontroler o nazwie Project. Na początku aranżujemy test, czyli wprowadzamy dane wejściowe. Następnie dokonujemy działania na metodzie, a na końcu sprawdzamy czy zwrócone wartości są zgodne z oczekiwanymi.

```
[Test]
public void ProjectViewTest()
{
    //Arrange
    var project = new ProjectsController(context);

    // Act
    var result = project.Details(new System.Guid("a1315678-589d-42d8-35d5-08d8ede1a239")) as ViewResult;

    // Assert
    Assert.AreEqual("Details", result.ViewName);
}
```

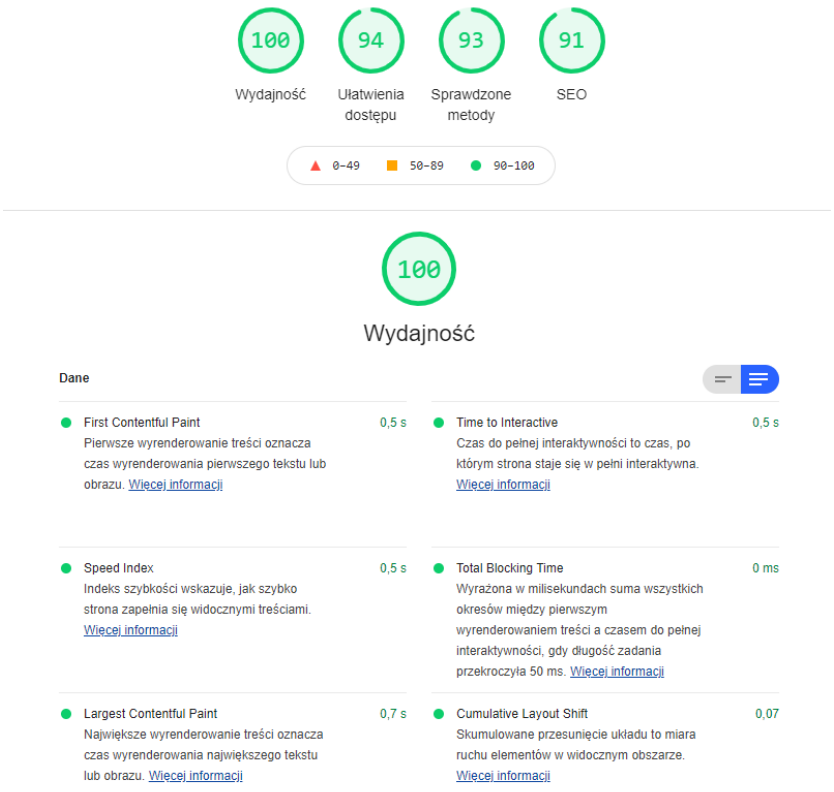
[21] Rafał Pawlak (2014). *Testowanie oprogramowania* wydawnictwo HELION, ISBN: 978-83-246-9308-5
[22] *Sylabus ISTQB – Poziom podstawowy (wersja 2011.1.3) > 2.Testowanie w cyklu życia oprogramowania > 2.2 Poziomy testów > 2.2.1 Testy modułowe* Odczytano dnia 25.03.2021 z http://getistqb.com/docs/sylabus-istqb-poziom-podstawowy/2-testowanie-w-cyklu-zycia-oprogramowania/2-2-poziomy-testow/2-2-1-testy-modulowe/
[23] *NUnit Documentation* Odczytano dnia 25.03.2021 z https://docs.nunit.org/

7. Podsumowanie i wnioski

W ramach prezentowanej pracy inżynierskiej powstał kompletny, zgodny ze wzorcem MVC oraz dobrymi praktykami, projekt konfiguratora mieszkań o nazwie „Housematica", w którego skład wchodzi część przeznaczona dla klienta, a także część przeznaczona dla administratora projektu.

Na część przeznaczoną dla klienta składa się aplikacja, której głównym założeniem przy projektowaniu była szybkość działania, spójność designu oraz łatwość w użytkowaniu. Na potrzeby wysokiej wydajności w ramach aplikacji, przebudowana została biblioteka Bootstrap, z której wyeliminowano nieużywane w ramach tego projektu bloki kodu, dzięki czemu strona zyskała na wydajności oraz czasie ładowania. Dodatkowo zwiększona została dostępność strony zgodnie z opublikowanymi zaleceniami[24].

Rys.7.1. Zrzut ekranu przedstawiający wynik testu części projektu Housematica przeznaczonej dla klienta za pomocą narzędzia Google Lighthouse [25]



[24] *Omówienie wymogów dostępności cyfrowej dla podmiotów publicznych* z dnia 15.02.2021 z https://www.gov.pl/web/dostepnosc-cyfrowa/omowienie-wymogow-dostepnosci-cyfrowej-dla-podmiotow-publicznych

[25] *Tools for Web Developers – Lighthouse* odczytano 25 marca 2020 z developers.google.com/web/tools/lighthouse

Część administracyjna to aplikacja, której głównym założeniem było stworzenie prostego procesu, pozwalającego na dodanie nowego konfiguratora bez posiadania wiedzy z dziedziny programowania czy szeroko rozumianego IT. Powyższy cel udało się zrealizować, dzięki czemu każda osoba, nawet taka, która nie posiada wiedzy specjalistycznej, będzie mogła w kilku krokach dodać nowy projekt, umieścić w ramach niego mieszkania oraz ich warianty, a finalnie udostępnić link do projektu klientom, aby mogli oni dokonywać konfiguracji. Ponadto panel administracyjny został wyposażony w dwie funkcjonalności ułatwiające późniejsze korzystanie z platformy:

- tworzenie zespołów, w ramach których zaproszone osoby mogą współdzielić administrację projektami oraz należącymi do nich mieszkaniami;
- przejrzyste statystyki dotyczące ilości wyświetleń projektów, unikalnych wyświetleń projektów oraz ilości stworzonych konfiguracji.

Na sam koniec pracy nad projektem, zarówno część aplikacji stworzonej dla klienta, jak i administratora, została dodatkowo skonfigurowana pod kątem wdrożenia jej na serwer aby w przyszłości w prosty sposób można było opublikować projekt w Internecie.

LITERATURA

Literatura zwarta:

1. Allen Sharon, Modelowanie danych, Helion, Warszawa 2006

2. Freeman Adam, Pro ASP.NET Core 3 Develop Cloud-Ready Web Applications Using MVC 3, Blazor, and Razor Pages Eight Edition, Apress, 2020

3. Khorikov Vladimir, Testy jednostkowe. Zasady, praktyki i wzorce, Helion, Warszawa 2020

4. Pawlak Rafał, Testowanie oprogramowania, Helion, Warszawa 2014, s.65 – s. 72

Internet:

1. Transakcje kupna i sprzedaży, https://stat.gov.pl/wyszukiwarka/?query=tag:transakcje+kupna+sprzeda%C5%BCy+nieruchomo%C5%9Bci (data odczytu 22.03.2021)

2. Customisation in practice: interactive product, https://programa.pl/en/2017/10/Customisation_in_practice_interactive_product_configurator (data odczytu 22.03.2021)

3. Junk, D. (2017). The 2 Psychological Principles that Make Product Configurators Such Stupidly Effective Ecommerce Tools, https://www.linkedin.com/pulse/2-psychological-principles-make-product-configurators-dennis-junk (data odczytu 22.03.2021)

4. Wikipedia.Wolna Encyklopedia. Hasło: Bootstrap (framework), https://pl.wikipedia.org/wiki/Bootstrap_(framework) (data odczytu 22.03.2021)

5. PopularitY of Programming Language, https://pypl.github.io/PYPL.html (data odczytu 22.03.2021)

6. Tung. L. (2020). Programming language popularity: JavaScript leads – 5 million new developers since 2017, https://www.zdnet.com/article/programming-language-popularity-javascript-leads-5-million-new-developers-since-2017/ (data odczytu 22.03.2021)

7. Institute of Computing Science, Poznan University of Technology AJAX w przykładach, http://www.cs.put.poznan.pl/jkobusinski/ajax.html (data odczytu 20.03.2021)

8. Wikipedia.Wolna Encyklopedia. Hasło: Model-view-controller, https://en.wikipedia.org/wiki/Model%E2%80%93view%2%80%93controller (data odczytu 20.03.2021)

9. Guid Struct, https://docs.microsoft.com/enus/dotnet/api/system.guid (data odczytu 24.03.2021)

10. WIT Wyższa Szkoła Informatyki Stosowanej i Zarządzania, Diagramy UML, http://info.wsisiz.edu.pl/~roksela/materialy/UML/diagramy_uml.pdf (data odczytu 24.03.2021)

11. Projektowanie i normalizacja bazy danych, https://www.sqlpedia.pl/projektowanie-i-normalizacja-bazy-danych/ (data odczytu 23.03.2021)

12. Tools for Web Developers – Lighthouse, developers.google.com/web/tools/lighthouse (data odczytu 25.03.2021)

13. What is WOFF2 and how do I enable it for my projects?, https://www.fonts.com/support/faq/what-is-woff2-and-how-do-i-enable-it (data odczytu 25.03.2021)

14. Building Shaders With Babylon.js, https://www.smashingmagazine.com/2016/11/building-shaders-with-babylon-js/ (data odczytu 25.03.2021)

15. What is Module Testing? Definition, Examples, https://www.guru99.com/module-testing.html (data odczytu 25.03.2021)

16. Sylabus ISTQB – Poziom podstawowy (wersja 2011.1.3) > 2.Testowanie w cyklu życia oprogramowania > 2.2 Poziomy testów > 2.2.1 Testy modułowe, http://getistqb.com/docs/sylabus-istqb-poziom-podstawowy/2-testowanie-w-cyklu-zycia-oprogramowania/2-2-poziomy-testow/2-2-1-testy-modulowe/ (data odczytu 25.03.2021)

17. NUnit Documentation, https://docs.nunit.org/ (data odczytu 25.03.2021)

18. Omówienie wymogów dostępności cyfrowej dla podmiotów publicznych, https://www.gov.pl/web/dostepnosc-cyfrowa/omowienie-wymogow-dostepnosci-cyfrowej-dla-podmiotow-publicznych (data odczytu 25.03.2021)

SPIS RYSUNKÓW