

**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ**

Szymon Matyka

Aplikacja webowa do zarządzania procesem najmu nieruchomości z obsługą dokumentacji i powiadomień

PRACA INŻYNIERSKA

dr inż. Grzegorz Drałus

Rzeszów, 2026

Spis treści

1. Wstęp	7
2. Analiza biznesowa projektu RentEase	8
2.1. Rynek najmu w Polsce	8
2.2. Cel projektu	8
2.3. Użytkownicy systemu	8
2.3.1. Persona: Janusz - Wynajmujący	9
2.3.2. Persona: Anna - Najemca	9
2.4. Kluczowe funkcjonalności	9
2.5. Analiza konkurencji	10
2.6. Analiza SWOT	11
2.7. Analiza Ryzyka Projektu	11
2.7.1. Ryzyka Technologiczne	11
2.7.2. Ryzyka Rynkowe i Biznesowe	12
2.7.3. Ryzyka Prawne i Regulacyjne	13
2.8. Model biznesowy i monetyzacja	13
2.9. Specyfikacja wymagań oprogramowania	14
3. Wykorzystane technologie i narzędzia	15
3.1. Język Python	15
3.2. Framework Django	15
3.3. PostgreSQL	16
3.4. Biblioteki pomocnicze (Pillow, WeasyPrint)	16
3.5. HTML, CSS, JavaScript i Bootstrap	16
4. Wygląd i struktura aplikacji	18
4.1. Strona główna	18
4.2. Logowanie i Rejestracja	19
4.3. Profil użytkownika	21
4.4. Szczegóły oferty	22
4.5. Tworzenie i edycja oferty	22
4.6. Panel Moje Oferty	24
4.7. Komunikator i lista konwersacji	24
4.8. Ulubione oferty	25

4.9.	Zarządzanie umowami	27
4.10.	Generator umów	27
4.11.	Podpisywanie umowy	28
5.	Dokumentacja techniczna projektu RentEase	30
5.1.	Architektura i wzorce projektowe	30
5.2.	Struktura bazy danych i modele	31
5.2.1.	Model LandlordUser	32
5.2.2.	Model TenantUser	33
5.2.3.	Model Offer	33
5.2.4.	Model Photo	34
5.2.5.	Model Conversation i Message	34
5.2.6.	Model Favorite	34
5.2.7.	Modele ContractTemplate i Contract	34
5.3.	Widoki i logika biznesowa	35
5.3.1.	Widoki uwierzytelniania	35
5.3.2.	Widoki ofert	36
5.3.3.	Widoki profili i ulubionych	36
5.3.4.	Szczegóły implementacji systemu komunikacji (Widoki konwersacji)	36
5.3.5.	Implementacja generatora umów PDF (Widoki umów)	37
5.4.	Formularze	39
5.4.1.	Formularz RegisterForm	40
5.4.2.	Formularz LoginForm	40
5.4.3.	Formularz ProfileForm	40
5.4.4.	Formularz OfferForm	40
5.4.5.	Formularz ContractPlaceholderForm	40
5.5.	Zarządzanie plikami statycznymi	41
5.6.	API	41
5.7.	Bezpieczeństwo systemu	42
5.8.	Testowanie oprogramowania (Unit Testing)	42
5.9.	Niestandardowe tagi szablonów	44
5.10.	Niestandardowe polecenia zarządzania	44
6.	Podsumowanie i wnioski	46

6.1. Kierunki dalszego rozwoju	46
Literatura	48

1. Wstęp

Celem pracy było stworzenie aplikacji webowej RentEase, która ułatwia i automatyzuje procesy związane z wynajmem nieruchomości. Głównym założeniem projektu było zbudowanie platformy, która usprawni komunikację i współpracę między właścicielami nieruchomości (landlordami) a osobami poszukującymi lokalu (tenantami).

Aplikacja została zrealizowana z wykorzystaniem nowoczesnych technologii webowych, z frameworkm Django [1] jako podstawą logiki. Wybór Django podyktowany był jego stabilnością, skalowalnością oraz bogatym ekosystemem, co pozwoliło na szybkie i bezpieczne wdrożenie zaawansowanych funkcjonalności. Za obsługę i przetwarzanie obrazów odpowiada biblioteka Pillow [2]. System został zaprojektowany z myślą o responsywności, aby zapewnić komfort użytkowania na różnych urządzeniach.

Praca opisuje cały cykl – od początkowej analizy biznesowej i badania rynku, przez projektowanie architektury systemu, implementację kluczowych modułów, aż po finalne testowanie i przygotowanie do wdrożenia. Niniejszy dokument został przygotowany w systemie L^AT_EX [3].

2. Analiza biznesowa projektu RentEase

Rozdział ten poświęcony jest analizie biznesowej aplikacji RentEase. Opisano w nim cele projektu, zdefiniowano grupy użytkowników, przedstawiono kluczowe funkcjonalności systemu, a także przeprowadzono analizę konkurencji i analizę SWOT. Wszystkie diagramy w pracy zostały stworzone w darmowym narzędziu diagrams.net [4], a logotyp aplikacji został wygenerowany z pomocą modelu AI Gemini [5].

2.1. Rynek najmu w Polsce

Rynek najmu w Polsce jest dynamicznym i rosnącym sektorem gospodarki. Jak wskazują raporty rynkowe z końca 2023 roku, po okresie gwałtownych wzrostów, sytuacja zaczęła się stabilizować, jednak popyt na mieszkania do wynajęcia wciąż pozostaje wysoki [6]. Średnie ceny najmu w dużych miastach utrzymują się na wysokim poziomie, z Warszawą jako liderem. Największym zainteresowaniem cieszą się mniejsze lokale, w tym mieszkania dwupokojowe, co stwarza niszę rynkową dla dedykowanych platform, takich jak RentEase.

2.2. Cel projektu

Głównym celem aplikacji RentEase jest usprawnienie i centralizacja procesu zarządzania najmem nieruchomości. Aplikacja ma za zadanie rozwiązać typowe problemy, takie jak brak efektywnej komunikacji między stronami, trudności w zarządzaniu ofertami oraz czasochłonne procesy związane z poszukiwaniem i wynajmowaniem nieruchomości. Dodatkowym, unikalnym celem jest integracja procesu generowania i podpisywania umów najmu bezpośrednio w platformie.

2.3. Użytkownicy systemu

System przewiduje dwa główne typy użytkowników. Poniżej przedstawiono ich szczegółowe persony.

2.3.1. Persona: Janusz - Wynajmujący

- **Wiek:** 45 lat
- **Zawód:** Przedsiębiorca
- **Potrzeby:** Janusz posiada kilka mieszkań na wynajem. Potrzebuje narzędzi, które pozwoli mu w łatwy sposób zarządzać ofertami, komunikować się z najemcami i automatyzować procesy związane z umowami. Ceni sobie oszczędność czasu i profesjonalizm.
- **Frustracje:** Obecnie korzysta z kilku różnych platform, co jest czasochłonne. Denerwuje go konieczność ręcznego przygotowywania umów i umawiania się na ich podpisanie.

2.3.2. Persona: Anna - Najemca

- **Wiek:** 28 lat
- **Zawód:** Programistka
- **Potrzeby:** Anna szuka nowoczesnego mieszkania w dobrej lokalizacji. Chce mieć możliwość szybkiego i łatwego przeglądania ofert, kontaktu z wynajmującym oraz bezpiecznego podpisania umowy online.
- **Frustracje:** Irytuja ją nieaktualne ogłoszenia i trudności w komunikacji z wynajmującymi. Obawia się skomplikowanych i niejasnych umów najmu.

2.4. Kluczowe funkcjonalności

Aplikacja RentEase oferuje szereg funkcjonalności, które mają na celu zaspokojenie potrzeb obu grup użytkowników:

- **Rejestracja i uwierzytelnianie użytkowników:** System umożliwia bezpieczne tworzenie kont i logowanie dla wynajmujących i najemców.
- **Zarządzanie ofertami:** Wynajmujący mają możliwość tworzenia, edytowania i usuwania ofert najmu.
- **Dodawanie zdjęć:** Aplikacja pozwala na łatwe dodawanie zdjęć do ofert, w tym za pomocą mechanizmu "przeciagnij i upuść"(drag-and-drop).
- **Szczegóły oferty:** Każda oferta prezentowana jest na dedykowanej stronie, zawierającej wszystkie niezbędne informacje.

- **Komunikacja:** Wbudowany komunikator pozwala na bezpośrednią i bezpieczną wymianę wiadomości między wynajmującym a potencjalnym najemcą.
- **Generowanie i podpisywanie umów:** Unikalna funkcjonalność pozwalająca na tworzenie szablonów umów, generowanie finalnych dokumentów z danymi użytkowników i oferty, oraz składanie podpisów online.
- **Responsywny design:** Interfejs aplikacji dostosowuje się do różnych rozmiarów ekranów, co zapewnia wygodne korzystanie zarówno na komputerach stacjonarnych, jak i urządzeniach mobilnych.

2.5. Analiza konkurencji

Rynek aplikacji do zarządzania najmem i portali z ogłoszeniami nieruchomości jest nasycony. Główne platformy działające w Polsce to m.in. Otodom, OLX Nieruchomości, Morizon. Platformy te oferują szeroką bazę ogłoszeń i zaawansowane wyszukiwarki.

Jednakże, większość z nich skupia się na aspekcie ogłoszeniowym, a proces finalizacji transakcji, w tym przygotowanie i podpisanie umowy, odbywa się poza platformą. RentEase wyróżnia się na tym tle poprzez integrację całego procesu w jednym miejscu. Podczas gdy istnieją oddzielne narzędzia do elektronicznego podpisywania dokumentów (np. DocuSign, Autenti), żadna z wiodących platform nieruchomości nie oferuje wbudowanego, w pełni zintegrowanego systemu do generowania umów najmu, co stanowi główną przewagę konkurencyjną projektu RentEase.

2.6. Analiza SWOT

Analiza SWOT (Strengths, Weaknesses, Opportunities, Threats) pozwala na ocenę strategiczną projektu.

Tabela 2.1. Analiza SWOT projektu RentEase

ANALIZA SWOT	
Mocne strony (Strengths)	Ślabe strony (Weaknesses)
Szanse (Opportunities)	Zagrożenia (Threats)
<ul style="list-style-type: none">- Innowacyjna funkcjonalność generowania i podpisywania umów- Kompleksowe rozwiązywanie "wszystko w jednym"- Intuicyjny i nowoczesny interfejs użytkownika- Zbudowana na stabilnej i bezpiecznej technologii (Django)	<ul style="list-style-type: none">- Mniejsza rozpoznawalność marki w porównaniu do dużych graczy- Ograniczona początkowo baza użytkowników i ofert- Potrzeba budowania zaufania do nowej platformy
<ul style="list-style-type: none">- Rosnące zapotrzebowanie na cyfryzację i automatyzację w branży nieruchomości- Możliwość integracji z innymi usługami (np. weryfikacja najemców, płatności online)- Ekspansja na rynki zagraniczne- Oferowanie usług premium dla agencji nieruchomości	<ul style="list-style-type: none">- Silna pozycja rynkowa istniejących portali ogłoszeniowych- Kwestie prawne i regulacyjne związane z umowami online- Szybkie zmiany technologiczne wymagające ciągłych aktualizacji

2.7. Analiza Ryzyka Projektu

Identyfikacja i analiza potencjalnych ryzyk jest kluczowym elementem w procesie zarządzania projektem informatycznym. Pozwala na wczesne przygotowanie strategii zaradczych i minimalizację negatywnego wpływu nieprzewidzianych zdarzeń na realizację celów projektu. Poniżej przedstawiono analizę głównych ryzyk zidentyfikowanych dla projektu RentEase, podzielonych na trzy kategorie.

2.7.1. Ryzyka Technologiczne

Ryzyka technologiczne są nieodłącznie związane z infrastrukturą, oprogramowaniem oraz procesem wytwórczym.

- **Ryzyko bezpieczeństwa danych:** Przechowywanie danych osobowych, w tym numerów PESEL i danych dowodów osobistych, stwarza ryzyko ich wycieku w wyniku ataku hakerskiego.
- *Strategia zaradcza:* Zastosowanie wbudowanych w Django mechanizmów bezpieczeństwa (ochrona przed CSRF, SQL Injection), regularne aktualizacje bi-

bliotek, szyfrowanie haseł, wdrożenie polityki silnych haseł oraz potencjalne szyfrowanie wrażliwych danych na poziomie bazy danych.

- **Ryzyko awarii i utraty danych:** Awaria serwera lub bazy danych może prowadzić do niedostępności usługi i trwałej utraty danych użytkowników, ofert i umów.
 - *Strategia zaradcza:* Wdrożenie regularnych, zautomatyzowanych kopii zapasowych bazy danych PostgreSQL. Wykorzystanie usług hostingowych zapewniających wysoki wskaźnik SLA (Service Level Agreement) oraz mechanizmy redundancji.
- **Ryzyko skalowalności:** Wraz ze wzrostem liczby użytkowników i ofert, aplikacja może przestać działać wydajnie, co objawia się długim czasem ładowania stron i generowania dokumentów.
 - *Strategia zaradcza:* Wybór skalowalnych technologii (Django, PostgreSQL). Optymalizacja zapytań do bazy danych, stosowanie mechanizmów cache'owania (np. Redis, Memcached) oraz możliwość wdrożenia load balancingu i skalowania horyzontalnego serwerów aplikacji w przyszłości.

2.7.2. Ryzyka Rynkowe i Biznesowe

Ta kategoria obejmuje ryzyka związane z otoczeniem konkurencyjnym i przyjęciem produktu przez rynek.

- **Ryzyko niskiej adopcji przez użytkowników:** Użytkownicy mogą nie być skłonni do zmiany swoich przyzwyczajeń i rezygnacji z popularnych, istniejących portali na rzecz nowej platformy.
 - *Strategia zaradcza:* Zaoferowanie unikalnej wartości (generator umów), która jest niedostępna u konkurencji. Intensywne działania marketingowe w mediach społecznościowych i grupach docelowych, a także wdrożenie modelu freemium, aby obniżyć barierę wejścia.
- **Ryzyko silnej odpowiedzi konkurencji:** Istniejący liderzy rynkowi (np. Odom, OLX) mogą skopiować kluczowe funkcjonalności RentEase, niwelując jej przewagę konkurencyjną.
 - *Strategia zaradcza:* Szybki rozwój produktu i budowanie lojalności wśród wczesnych użytkowników. Skupienie się na doskonałej obsłudze klienta i cią-

głe wprowadzanie innowacji, aby utrzymać pozycję lidera w niszy kompleksowego zarządzania najmem.

2.7.3. Ryzyka Prawne i Regulacyjne

Projekty przetwarzające dane osobowe i związane z dokumentacją prawną muszą uwzględniać otoczenie regulacyjne.

- **Ryzyko związane z RODO (GDPR):** Nieprawidłowe przetwarzanie lub bezpieczenstwo danych osobowych może prowadzić do wysokich kar finansowych.
 - *Strategia zaradcza:* Ścisłe przestrzeganie zasad RODO, w tym prawa do bycia zapomnianym, minimalizacji danych i transparentności. Opracowanie i udostępnienie użytkownikom klarownej polityki prywatności.
- **Ryzyko prawnej ważności umów online:** W niektórych sytuacjach prawnych lub w przypadku sporów sądowych, ważność umowy podpisanej za pomocą prostego podpisu elektronicznego (kliknięcie "akceptuję") może być kwestionowana.
 - *Strategia zaradcza:* Wyraźne informowanie użytkowników o charakterze składanego oświadczenia woli. W przyszłości możliwa jest integracja z dostawcami kwalifikowanych podpisów elektronicznych (np. Autenti, DocuSign), co zapewni pełną moc prawną dokumentom.

2.8. Model biznesowy i monetyzacja

Model biznesowy aplikacji RentEase opiera się na strategii freemium. Podstawowe funkcjonalności, takie jak przeglądanie ofert i komunikacja, są dostępne dla wszystkich użytkowników bezpłatnie.

Monetyzacja planowana jest poprzez wprowadzenie planów subskrypcyjnych dla wynajmujących, oferujących zaawansowane funkcjonalności:

- **Plan Premium:** Nielimitowana liczba aktywnych ofert, promowanie ogłoszeń, dostęp do zaawansowanych statystyk oraz nielimitowana możliwość generowania i podpisywania umów.
- **Opłaty transakcyjne:** Możliwość wprowadzenia niewielkiej opłaty za każdą wygenerowaną i podpisaną umowę w planie darmowym.

2.9. Specyfikacja wymagań oprogramowania

Kluczowym elementem analizy biznesowej jest zdefiniowanie wymagań, które musi spełnić gotowy system. Podzielono je na dwie kategorie.

Wymagania funkcjonalne (cechy i zachowania systemu):

- System musi umożliwiać założenie konta użytkownika z podziałem na role: Wynajmujący (Landlord) i Najemca (Tenant).
- Wynajmujący musi posiadać możliwość tworzenia, edycji i usuwania ofert wynajmu nieruchomości wraz z załączoną galerią zdjęć.
- Najemca musi mieć możliwość przeszukiwania ofert oraz dodawania ich do listy "Ulubionych".
- System musi realizować asynchroniczną wymianę wiadomości tekstowych pomiędzy Najemcą a Wynajmującym w kontekście konkretnej oferty.
- System musi udostępniać mechanizm pozwalający Wynajmującemu na wygenerowanie spersonalizowanej umowy najmu w formacie PDF na podstawie zdefiniowanego wcześniej szablonu.

Wymagania niefunkcjonalne (jakość, bezpieczeństwo i wydajność):

- Hasła użytkowników nie mogą być przetrzymywane w bazie danych w postaci jawnej (plain-text). Wymagane jest użycie silnych algorytmów haszujących (np. PBKDF2).
- Interfejs aplikacji musi w pełni obsługiwać wytyczne Responsive Web Design, umożliwiając komfortowe korzystanie na ekranach o szerokości od 320px wzwyż.
- Czas generowania pliku PDF z umową nie powinien przekraczać 5 sekund przy standardowym obciążeniu serwera.
- Aplikacja musi implementować ochronę przed atakami typu CSRF (Cross-Site Request Forgery) na wszystkich formularzach przesyłających dane (metody POST, PUT, DELETE).

3. Wykorzystane technologie i narzędzia

Wybór odpowiedniego stosu technologicznego (ang. *tech stack*) jest kluczowym etapem projektowania każdego systemu informatycznego. Zastosowane narzędzia muszą spełniać wymagania wydajnościowe, zapewniać wysoki poziom bezpieczeństwa oraz umożliwiać łatwe skalowanie i utrzymanie kodu w przyszłości. W niniejszym rozdziale zaprezentowano szczegółową charakterystykę technologii wykorzystanych do realizacji projektu RentEase.

3.1. Język Python

Python to wszechstronny język programowania wysokiego poziomu o ogólnym przeznaczeniu. Został wybrany jako główny język implementacji logiki serwerowej ze względu na swoją bardzo czytelną składnię, ogromny ekosystem bibliotek oraz doskonałe wsparcie dla paradygmatu programowania obiektowego. Język ten jest interpretowany, co oznacza, że kod źródłowy nie wymaga uprzedniej kompilacji do postaci pliku wykonywalnego, lecz jest przetwarzany w czasie rzeczywistym przez interpreter. Pozwala to na niezwykle szybkie prototypowanie (ang. *Rapid Application Development*) oraz ułatwia testowanie poszczególnych modułów.

3.2. Framework Django

Django to darmowy, otwartoźródłowy framework napisany w języku Python, służący do tworzenia zaawansowanych, bezpiecznych i łatwych w utrzymaniu aplikacji webowych. Jego architektura opiera się na wzorcu MVT (Model-View-Template). Główną zaletą Django jest rygorystyczne przestrzeganie reguły "Don't Repeat Yourself" (DRY) oraz konwencja "batteries included", co oznacza dostarczenie programiście wielu gotowych, przetestowanych modułów już na starcie.

W ramach realizacji systemu RentEase, framework Django dostarczył:

- **System autoryzacji i uwierzytelniania** – gotowe mechanizmy szyfrowania haseł (PBKDF2) oraz zarządzania sesjami użytkowników.
- **Panel administratora** – automatycznie generowany interfejs graficzny pozwalający na zarządzanie danymi w bazie bez konieczności pisania zapytań SQL.

- **System mapowania obiektowo-relacyjnego (ORM)** – potężne narzędzie pozwalające na tworzenie struktury bazy danych za pomocą klas języka Python.

3.3. PostgreSQL

Jako główny system zarządzania relacyjną bazą danych (RDBMS) wybrano PostgreSQL. Jest to jedno z najbardziej zaawansowanych, otwartoźródłowych rozwiązań na rynku. W przeciwieństwie do domyślnie konfigurowanego w Django silnika SQLite, PostgreSQL doskonale radzi sobie ze współbieżnością zapytań pochodzących od wielu użytkowników jednocześnie. Zapewnia pełną zgodność z właściwościami ACID (Atomicity, Consistency, Isolation, Durability), co jest kluczowe w systemach przetwarzających dane umów oraz informacje osobiste użytkowników.

3.4. Biblioteki pomocnicze (Pillow, WeasyPrint)

Do zrealizowania specyficznych funkcji biznesowych systemu użyto kilku dedykowanych bibliotek:

- **Pillow** – zaawansowana biblioteka do przetwarzania obrazów w Pythonie. Została wykorzystana w systemie do obsługi mechanizmu dodawania zdjęć do ofert. Zapewnia automatyczne skalowanie, kompresję i zapisywane zdjęcia wgrywanych przez wynajmujących, co optymalizuje czas ładowania strony.
- **ReportLab** – biblioteka wykorzystana w innowacyjnym module generatora umów. Pozwala na programistyczne tworzenie dokumentów w formacie PDF, które można następnie cyfrowo podpisać.

3.5. HTML, CSS, JavaScript i Bootstrap

Warstwa prezentacji (ang. *frontend*) aplikacji RentEase została zbudowana przy wykorzystaniu klasycznych, sprawdzonych technologii webowych. HTML5 odpowiada za semantyczną strukturę dokumentu, podczas gdy CSS3 został użyty do nadania estetycznego wyglądu. Do zapewnienia pełnej responsywności (RWD - *Responsive Web Design*) zastosowano framework Bootstrap [7], który dzięki systemowi siatek (grid system) gwarantuje poprawne wyświetlanie interfejsu zarówno na ekranach monitorów, jak i urządzeniach mobilnych. Logika interaktywna na stronie klienta (np. filtrowanie

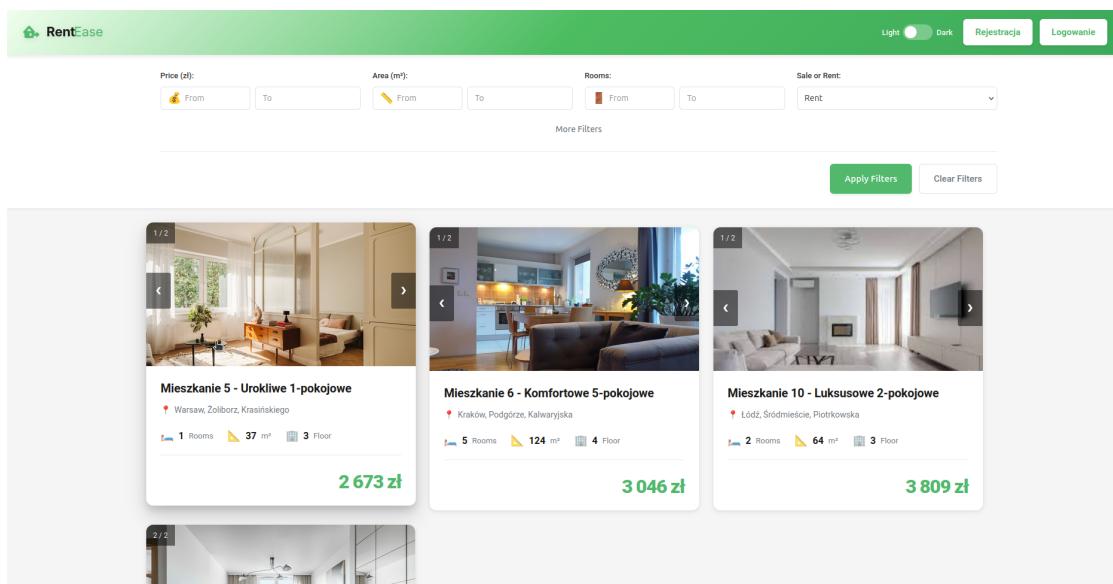
i mechanizm przeciągnij i upuść) opiera się na języku JavaScript z użyciem API Fetch do asynchronicznego pobierania danych.

4. Wygląd i struktura aplikacji

Rozdział ten przedstawia wygląd kluczowych ekranów aplikacji RentEase. Dla każdego widoku zamieszczono opis jego funkcjonalności oraz placeholder na zrzut ekranu.

4.1. Strona główna

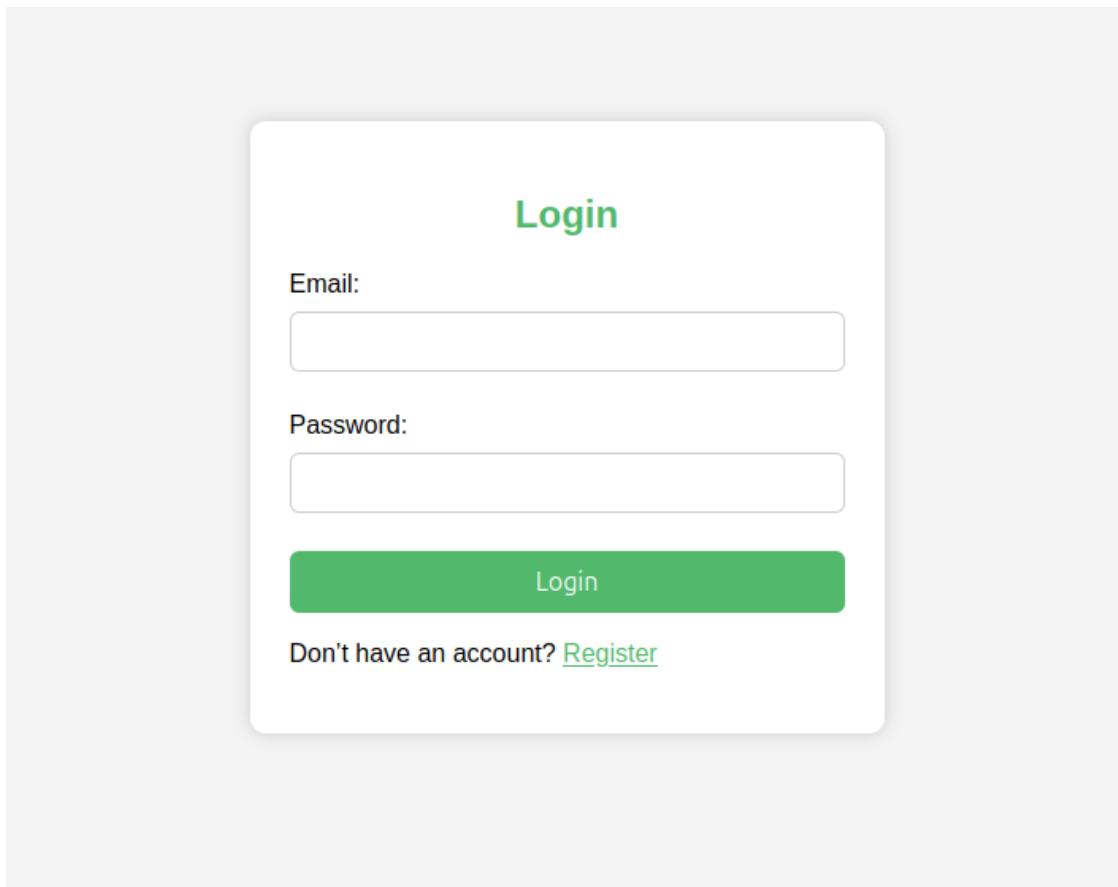
Główny widok aplikacji, obsługiwany przez widok ‘mainSite’, dynamicznie generuje listę ofert pobranych z bazy danych. Domyślnie oferty są sortowane malejąco według daty dodania, aby promować najnowsze ogłoszenia. Zastosowano mechanizm paginacji, który dzieli wyniki na strony po 10 ofert, co optymalizuje czas ładowania i poprawia wydajność. Logika wyszukiwania i filtrowania jest realizowana po stronie serwera. Parametry z formularza (takie jak lokalizacja, cena czy liczba pokoi) są przesyłane metodą GET, a widok Django filtry obiekty ‘Offer’ za pomocą metody ‘filter()’ na query-set’cie. W górnej części strony znajduje się pasek nawigacyjny, którego zawartość (np. linki do profilu lub panelu "Moje Oferty") jest dynamicznie renderowana w zależności od statusu uwierzytelnienia użytkownika i jego roli (Wynajmujący/Najemca) przechowywanej w sesji.



Rys. 4.1. Widok strony głównej z listą ofert.

4.2. Logowanie i Rejestracja

Proces uwierzytelniania i tworzenia konta jest obsługiwany przez dedykowane widoki ‘loginPanel’ i ‘registerPanel’. Wykorzystują one predefiniowane w pliku ‘forms.py’ klasy formularzy: ‘LoginForm’ oraz ‘RegisterForm’. Formularze te implementują logikę walidacji danych, taką jak sprawdzanie unikalności adresu e-mail czy zgodności haseł, zarówno po stronie serwera, jak i opcjonalnie po stronie klienta z użyciem atrybutów HTML5. W przypadku próby logowania, podane hasło jest porównywane z hashem przechowywanym w bazie danych przy użyciu funkcji ‘check_password()’ z frameworka Django. Po pomyślnym uwierzytelnieniu, identyfikator użytkownika oraz jego typ (‘landlord’ lub ‘tenant’) są zapisywane w sesji serwerowej, co pozwala na identyfikację użytkownika przy kolejnych żądaniach. Sesja jest zarządzana przez wbudowany w Django mechanizm oparty na ciasteczkach.



Rys. 4.2. Formularz logowania.

The image shows a registration form titled "Register". It features two main buttons at the top: "Landlord" (green) and "Tenant" (white). Below these are four input fields: "Name", "Surname", "Email", and "Password". A large green "Register" button is positioned below the inputs. At the bottom, there is a link "Already have an account? [Login](#)".

Register

Landlord Tenant

Name

Surname

Email

Password

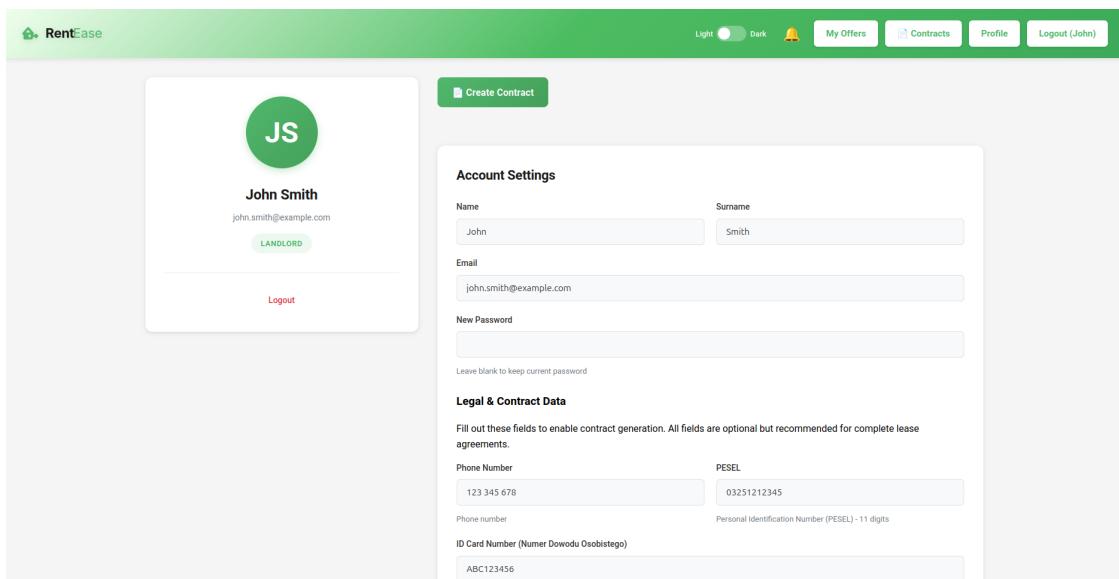
Register

Already have an account? [Login](#)

Rys. 4.3. Formularz rejestracji.

4.3. Profil użytkownika

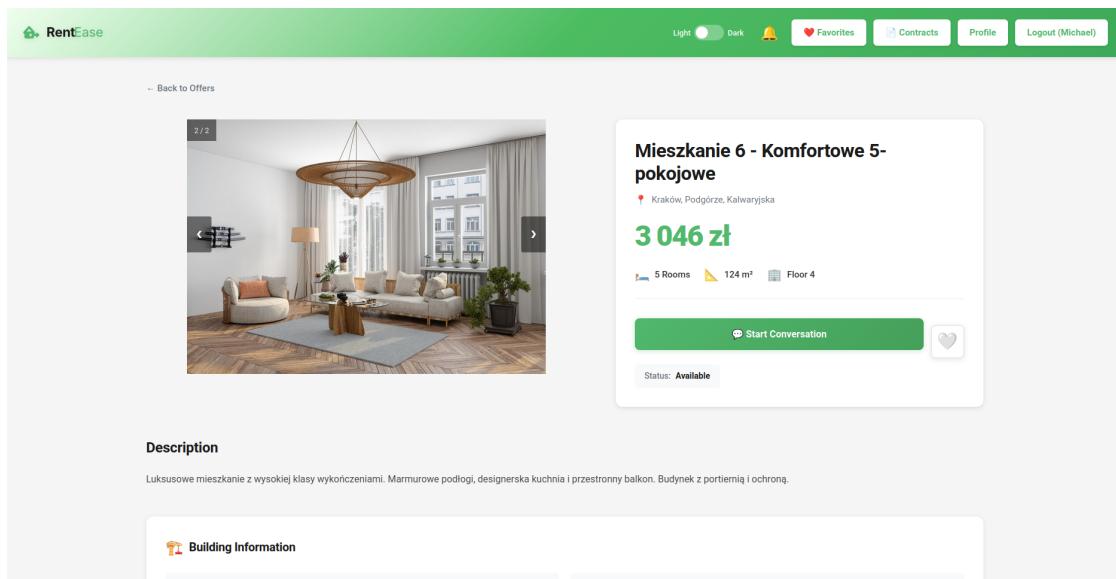
Panel profilu użytkownika jest obsługiwany przez widok ‘profile’, który jest chroniony dekoratorem ‘@login_required’. Przy żądaniu GET, widok inicjalizuje formularz ‘ProfileForm’ danymi zalogowanego użytkownika, pobierając je z bazy i wstępnie wypełniając pola formularza. Dzięki temu użytkownik widzi swoje aktualne dane. Zmiana danych odbywa się poprzez wysłanie żądania POST. Dane z formularza są ponownie walidowane (np. poprawność formatu numeru PESEL). Jeśli dane są prawidłowe, metoda ‘save()’ formularza aktualizuje odpowiedni obiekt ‘LandlordUser’ lub ‘TenantUser’ w bazie danych. Funkcjonalność zmiany hasła jest zazwyczaj odseparowana i dla bezpieczeństwa wymaga podania starego hasła w celu weryfikacji tożsamości.



Rys. 4.4. Strona profilu użytkownika.

4.4. Szczegóły oferty

Po kliknięciu w ofertę, użytkownik zostaje przekierowany na stronę ze szczegółami. Widok ten zawiera wszystkie informacje o nieruchomości, galerię zdjęć, lokalizację na mapie oraz przycisk umożliwiający rozpoczęcie konwersacji z wynajmującym.



Rys. 4.5. Strona szczegółów oferty.

4.5. Tworzenie i edycja oferty

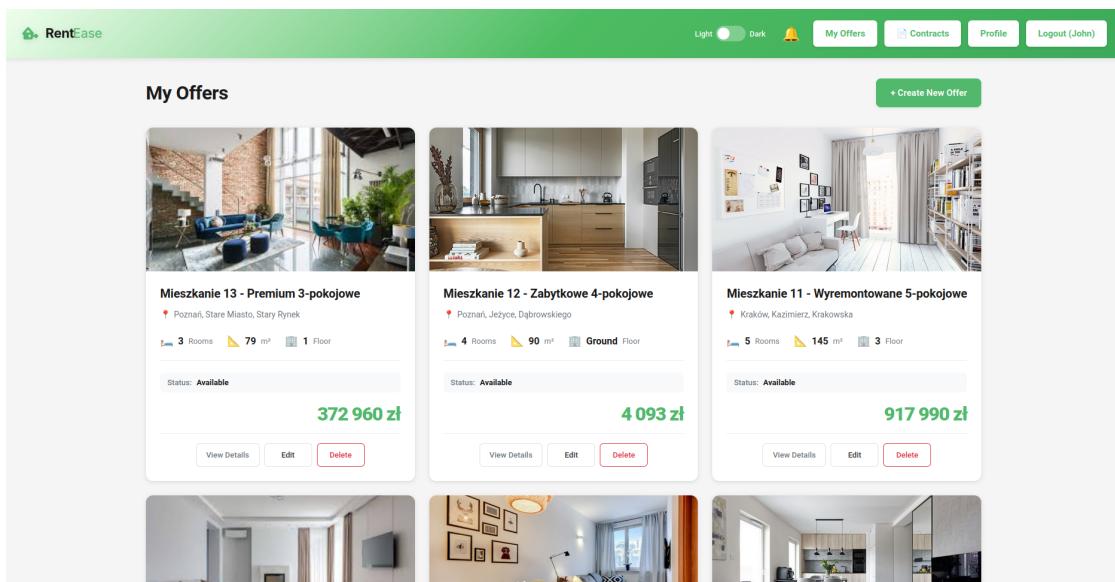
Logika tworzenia i edycji ofert jest często zunifikowana w jednym widoku (np. ‘manage_offer’), który rozróżnia operacje na podstawie obecności parametru ‘offer_id’ w URL. W przypadku edycji, formularz ‘OfferForm’ (będący klasą typu ‘ModelForm’) jest inicjalizowany istniejącą instancją obiektu ‘Offer’. Dla nowej oferty, formularz jest pusty. Po przesłaniu metodą POST, dane są walidowane. Jeśli formularz jest poprawny, jest on zapisywany do bazy danych. Kluczowe jest, że pole ‘user’ (właściciel oferty) nie jest częścią formularza widoczną dla użytkownika, lecz jest automatycznie przypisywane w widoku na podstawie zalogowanego użytkownika (‘request.user’), co zapobiega podszywaniu się. Obsługa wgrywania wielu plików (zdjęć) jest realizowana poprzez specjalne pole formularza ‘FileField’ z atrybutem ‘multiple’ i wymaga odpowiedniej obsługi pętli w widoku do przetworzenia każdego pliku z ‘request.FILES’.

The screenshot shows the 'Create Offer' page on the RentEase platform. At the top, there's a green header bar with the RentEase logo, a light/dark mode toggle, and user navigation links for 'My Offers', 'Contracts', 'Profile', and 'Logout (John)'. The main form area has a white background and is titled 'Create Offer'. It contains several sections: 'Basic Information' with fields for 'Title' (a text input field) and 'Description' (a large text area); 'Status' (a dropdown menu showing 'Available') and 'Sale or Rent' (another dropdown menu showing 'Rent'); 'Photos' (a section with a dashed box for dragging images, featuring a camera icon and the text 'Drag and drop images here or click to select from your computer'); and 'Location' (a small, partially visible section). The entire form is contained within a light gray box.

Rys. 4.6. Formularz tworzenia/edykcji oferty.

4.6. Panel Moje Oferty

Panel "Moje Oferty" to widok chroniony, dostępny wyłącznie dla użytkowników z rolą "Wynajmujący". Jego głównym zadaniem jest wyświetlenie spersonalizowanej listy ofert. Widok 'my_offers' wykonuje zapytanie do bazy danych, filtrując obiekty 'Offer' gdzie pole 'user' jest równe aktualnie zalogowanemu użytkownikowi ('Offer.objects.filter(user=request.user)'). Tabela z ofertami zawiera linki do edycji (prowadzi do widoku 'manage_offer' z ID oferty) oraz przyciski do usuwania. Operacja usuwania jest realizowana przez dedykowany widok 'delete_offer', który dla bezpieczeństwa powinien akceptować wyłącznie żądania POST lub DELETE, aby zapobiec przypadkowemu usunięciu oferty poprzez wejście na link (żądanie GET). Zmiana statusu oferty (np. na "Zarezerwowana") może być realizowana asynchronicznie za pomocą dedykowanego endpointu API.

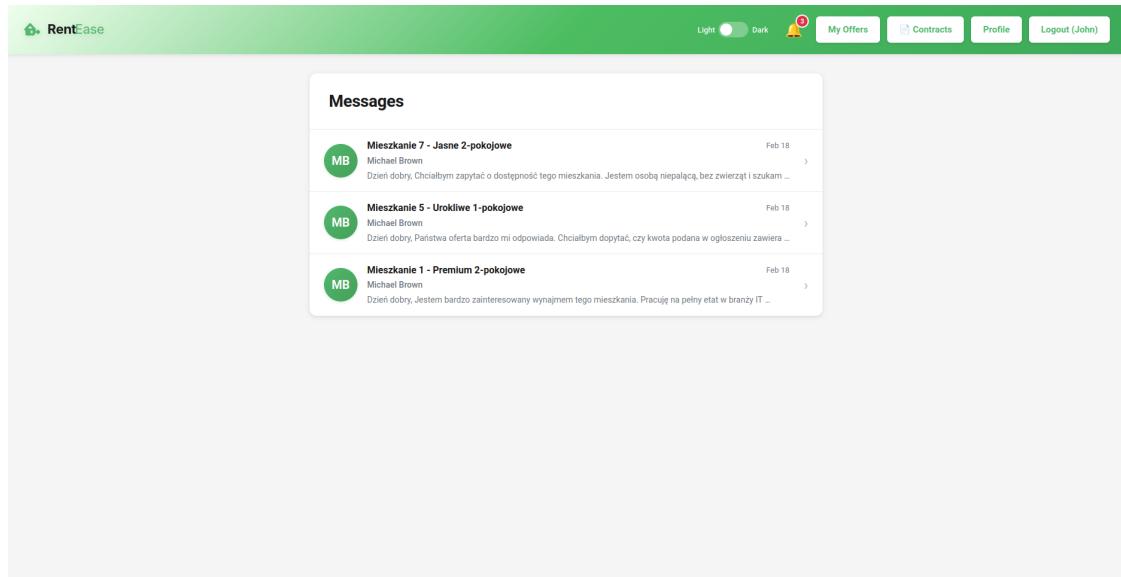


Rys. 4.7. Panel z listą ofert wynajmującego.

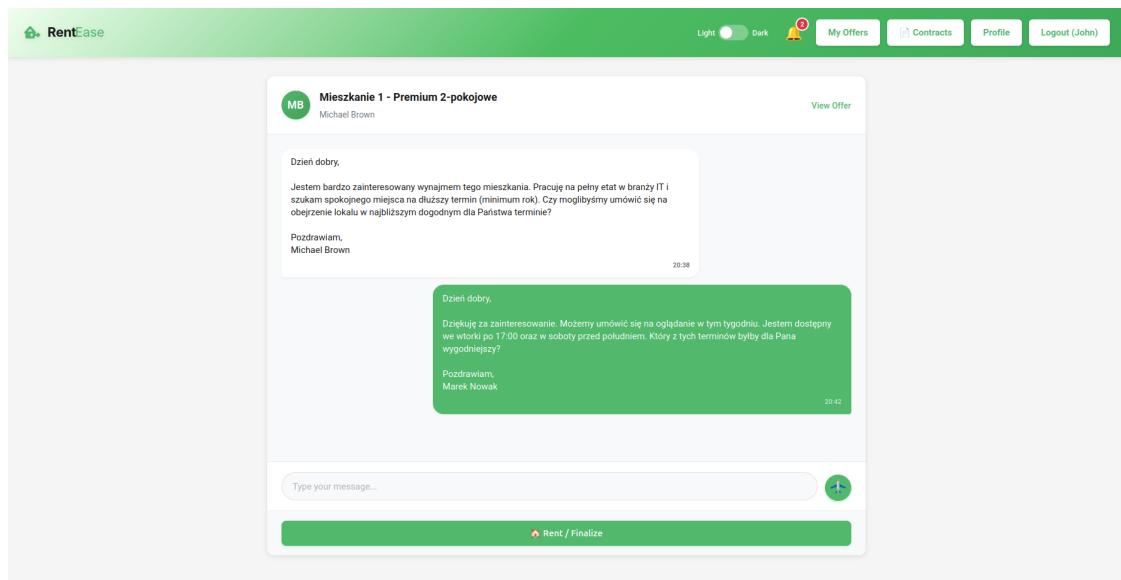
4.7. Komunikator i lista konwersacji

System komunikacji opiera się na dwóch głównych modelach: 'Conversation', który łączy Wynajmującego, Najemcę i Ofertę, oraz 'Message', przechowujący treść wiadomości i jej metadane. Widok listy konwersacji ('conversations_list') pobiera wszystkie obiekty 'Conversation', w których zalogowany użytkownik występuje jako 'tenant' lub 'landlord'. Używa się do tego złożonego zapytania z warunkiem 'OR' (w Django 'Q' objects: 'Conversation.objects.filter(Q(tenant=user) | Q(landlord=user))'). Widok szczegółów konwersacji 'conversation_detail' jest zabezpieczony w celu zapewnienia, że

tylko uczestnicy rozmowy mają do niej dostęp. Wysyłanie nowej wiadomości odbywa się poprzez żądanie POST do tego widoku, co tworzy nowy obiekt ‘Message’ powiązany z daną konwersacją.



Rys. 4.8. Lista konwersacji.



Rys. 4.9. Widok konwersacji między użytkownikami.

4.8. Ulubione oferty

Funkcjonalność "Ulubionych" jest zaimplementowana za pomocą modelu ‘Favorite’, który tworzy relację wiele-do-wielu pomiędzy użytkownikami (‘TenantUser’) a ofertami (‘Offer’). Model ten przechowuje pary ‘(tenant_id, offer_id)’. Strona z listą ulubionych

ofert pobiera wszystkie obiekty ‘Favorite’ dla zalogowanego najemcy, a następnie, poprzez relacje, wyciąga z bazy powiązane z nimi obiekty ‘Offer’.

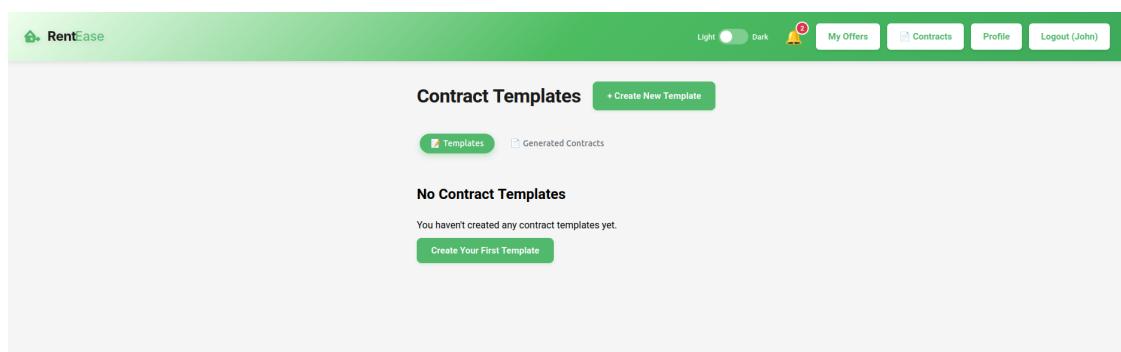
The screenshot shows the RentEase mobile application interface. At the top, there is a green header bar with the RentEase logo, a light/dark mode switch, and navigation links for Favorites, Contracts, Profile, and Logout (Michael). Below the header, the title "My Favorites" is displayed. Three apartment listings are shown in a grid:

- Mieszkanie 13 - Premium 3-pokojowe**
Poznań, Stare Miasto, Stary Rynek
3 Rooms, 79 m², 1 Floor
372 960 zł
- Mieszkanie 3 - Współczesne 1-pokojowe**
Wrocław, Stare Miasto, Rynek
1 Room, 30 m², 2 Floor
563 822 zł
- Mieszkanie 1 - Premium 2-pokojowe**
Kraków, Podgórze, Kalwaryjska
2 Rooms, 63 m², 4 Floor
507 650 zł

Rys. 4.10. Lista ulubionych ofert.

4.9. Zarządzanie umowami

Sekcja zarządzania umowami stanowi centrum kontroli dla wynajmującego, pozwalając na przygotowanie i standaryzację dokumentacji najmu. Zamiast tworzyć umowy od zera dla każdego najemcy, wynajmujący może zdefiniować uniwersalne szablony. Szablon taki zawiera stałe elementy umowy, takie jak ogólne warunki najmu, obowiązki stron czy klauzule dotyczące rozwiązania umowy. W treści szablonu umieszcza się specjalne znaczniki (placeholders), np. {{tenant_name}} czy {{rent_price}}, które podczas finalnego generowania dokumentu zostaną automatycznie podmienione na konkretne dane najemcy i oferty. Takie podejście drastycznie skraca czas potrzebny na przygotowanie umowy i minimalizuje ryzyko pomyłek.



Rys. 4.11. Panel zarządzania szablonami umów.

4.10. Generator umów

Kreator szablonu umowy to interaktywne narzędzie, które umożliwia wynajmującemu zdefiniowanie struktury i treści dokumentu. Interfejs kreatora pozwala na swobodne formatowanie tekstu oraz wstawianie dynamicznych pól (placeholderów) z predefiniowanej listy. Lista ta jest pogrupowana tematycznie i zawiera wszystkie dane, które system może automatycznie uzupełnić, np. dane osobowe wynajmującego i najemcy, szczegóły oferty (adres, cena, powierzchnia) czy daty. Po stworzeniu szablonu, proces generowania umowy dla konkretnego najemcy jest niezwykle prosty. Wynajmujący wybiera odpowiedniego najemcę z listy konwersacji, aplikacja automatycznie przypisuje ofertę, której dotyczyła rozmowa, a następnie wystarczy wybrać jeden z wcześniej przygotowanych szablonów. System sam wypełnia wszystkie luki w dokumencie, prezentując gotową umowę do oglądu.

Select Offers for This Template
Check the offers you want to assign this contract template to. You can select multiple offers.

Tenant Data

Tenant Name
[Will be filled when contract is gene...]

Tenant Surname
[Will be filled when contract is gene...]

Full Name
[Will be filled when contract is gene...]

PESEL
[Will be filled when contract is gene...]

ID Card Number
[Will be filled when contract is gene...]

Address
[Will be filled when contract is gene...]

Phone
[Will be filled when contract is gene...]

Email
[Will be filled when contract is gene...]

Bank Account
[Will be filled when contract is gene...]

Assign to Offers:

Mieszkanie 13 - Premium 3-pokojowe
Poznań, Stare Miasto, Stary Rynek

Mieszkanie 12 - Zabytkowe 4-pokojowe
Poznań, Jeżyce, Dąbrowskiego

Mieszkanie 11 - Wyremontowane 5-pokojowe

Niniejsza umowa zostaje zawarta pomiędzy Wynajmującym, posługującym się adresem e-mail [Email](#) oraz numerem telefonu [Phone](#), a Naiemcą, który jest [Tenant Name](#) [PESEL](#).

§1 Przedmiot Umowy
Wynajmujący oddaje w najem lokal mieszkalny o nazwie, znajdujący się w lokalizacji [Location](#). Mieszkanie ma powierzchnię [Area](#) i składa się z [Rooms](#) pokoi, a znajduje się na kondygnacji: [Floor](#).

§2 Oplaty i Płatności

- Naiemca zobowiązuje się płacić miesięczny czynsz w wysokości [Price](#).
- Płatności będą dokonywane przelewem na rachunek bankowy Wynajmującego o numerze: [Bank Account](#)

§3 Postanowienia końcowe

[Preview](#) [Save Template](#)

Caro renumier ABC123456

Address
ul. Marszałkowska 10/15, 00-001, W...

Phone
123 345 678

Email
john.smith@example.com

Bank Account
12 1020 1013 0000 0102 0345 6789

Property Data

Property Title
[From selected offer]

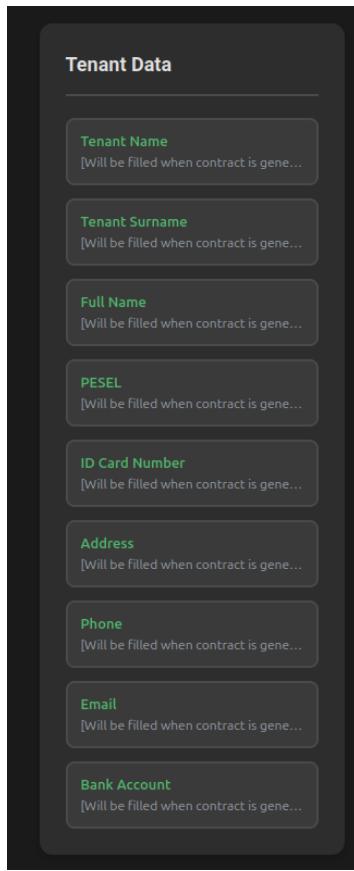
Location
[From selected offer]

Price
[From selected offer]

Area
[From selected offer]

Rooms
[From selected offer]

Rys. 4.12. Kreator szablonu umowy.

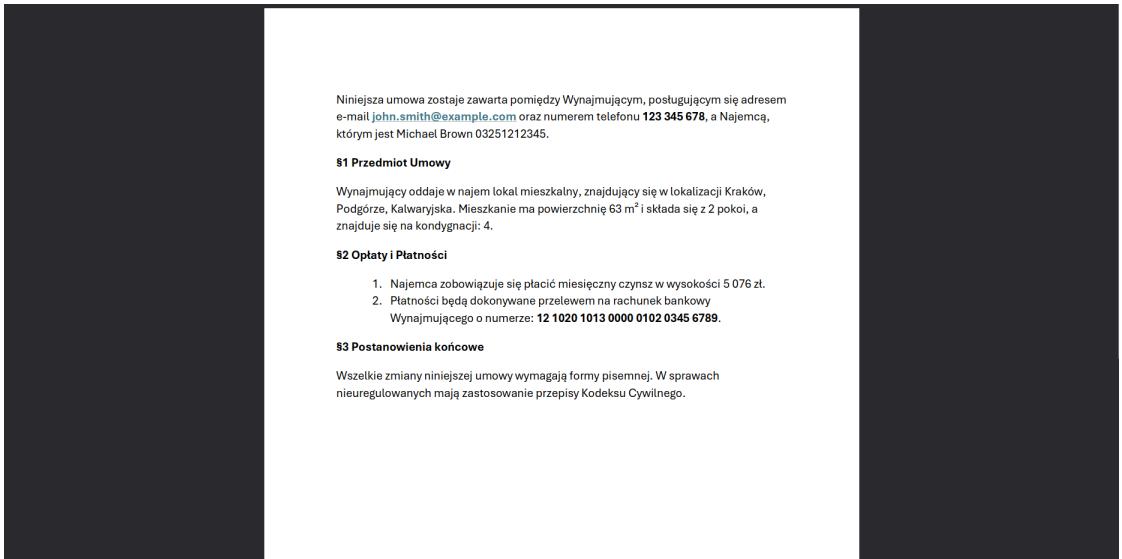


Rys. 4.13. Widok listy przykładowych zmiennych (placeholderów) dostępnych w kreatorze.

4.11. Podpiswanie umowy

Finalnym etapem procesu jest podpisanie umowy. Po wygenerowaniu dokumentu przez wynajmującego, naiemca otrzymuje powiadomienie w aplikacji o nowej umowie ocze-

kującej na akceptację. Obie strony mają dostęp do tego samego, finalnego dokumentu w dedykowanym widoku. Mogą go dokładnie przejrzeć, upewniając się, że wszystkie dane są poprawne. Proces podpisu jest realizowany poza aplikacją przez np.: profil zaufany. System rejestruje datę i godzinę złożenia podpisu przez każdą ze stron. Gotowy, podpisany dokument jest archiwizowany w systemie i dostępny do pobrania w formacie PDF dla obu stron w dowolnym momencie.

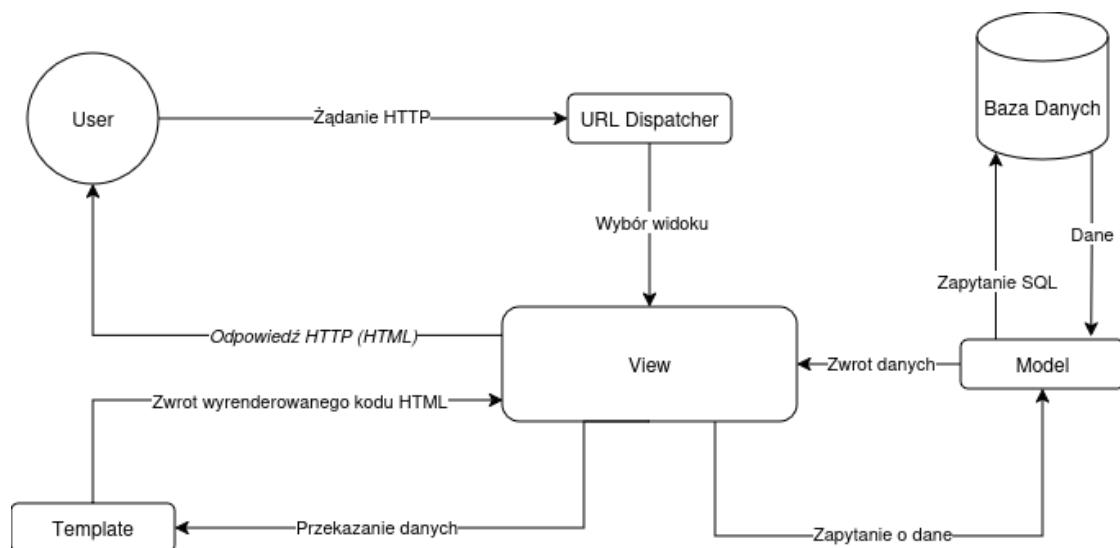


Rys. 4.14. Widok umowy gotowej do podpisania.

5. Dokumentacja techniczna projektu RentEase

5.1. Architektura i wzorce projektowe

Aplikacja RentEase została zaimplementowana w oparciu o architekturę Model-View-Template (MVT), która jest charakterystyczna dla frameworka Django. Jest to wariacja na temat popularnego wzorca Model-View-Controller (MVC).



Rys. 5.1. Diagram architektury MVT w Django.

- **Model (Model)**: Odpowiada za strukturę danych i logikę biznesową. W Django, modele są klasami Pythona, które mapują się na tabele w bazie danych. Definiują one pola i zachowania przechowywanych danych.
- **View (Widok)**: W Django, widok jest odpowiedzialny za przyjmowanie żądań HTTP i zwracanie odpowiedzi. Widok komunikuje się z modelem, aby pobrać dane, a następnie przekazuje je do szablonu.
- **Template (Szablon)**: Jest to warstwa prezentacji. Szablony w Django to pliki HTML z osadzonym specjalnym językiem szablonów Django (Django Template Language, DTL), który pozwala na dynamiczne generowanie treści.

Oprócz głównego wzorca MVT, w projekcie można zidentyfikować inne wzorce projektowe:

- **Singleton (Singleton)**: Django samo w sobie implementuje wzorzec Singleton w wielu miejscach, na przykład w przypadku ustawień projektu (`settings.py`), które są ładowane raz i dostępne globalnie w całej aplikacji.

- **Decorator (Dekorator):** Dekoratory są szeroko stosowane w Django do modyfikacji zachowania funkcji lub klas. W RentEase, dekoratory takie jak `@login_required` są używane do zabezpieczania widoków, które wymagają zalogowanego użytkownika.
- **Factory (Fabryka):** Chociaż nie jest to jawnie zaimplementowane, formularze Django, zwłaszcza `ModelForm`, działają na podobnej zasadzie co fabryki, tworząc obiekty formularzy na podstawie definicji modeli.

5.2. Struktura bazy danych i modele

Baza danych została zaprojektowana w sposób relacyjny. Modele Django, będące klasami w języku Python, odwzorowują tabele i relacje w bazie. Na Listingu 5.1 przedstawiono kod źródłowy kluczowego dla systemu modelu ‘Offer’, który reprezentuje pojedynczą ofertę nieruchomości.

```

1 from django.db import models
2 from .users import LandlordUser, TenantUser
3
4 class Offer(models.Model):
5     STATUS_CHOICES = [
6         ('Available', 'Dostępna'),
7         ('Reserved', 'Zarezerwowana'),
8         ('Rented', 'Wynajta'),
9     ]
10
11     title = models.CharField(max_length=255, verbose_name="Tytuł")
12     body = models.TextField(verbose_name="Opis")
13     user = models.ForeignKey(LandlordUser, on_delete=models.CASCADE, related_name='offers')
14     status = models.CharField(max_length=10, choices=STATUS_CHOICES, default='Available')
15
16     location = models.CharField(max_length=255, verbose_name="Lokalizacja")
17     price = models.DecimalField(max_digits=10, decimal_places=2, verbose_name="Cena")
18
19     area = models.DecimalField(max_digits=7, decimal_places=2, verbose_name="Powierzchnia")
20     number_of_rooms = models.PositiveIntegerField(verbose_name="Liczba pokoi")

```

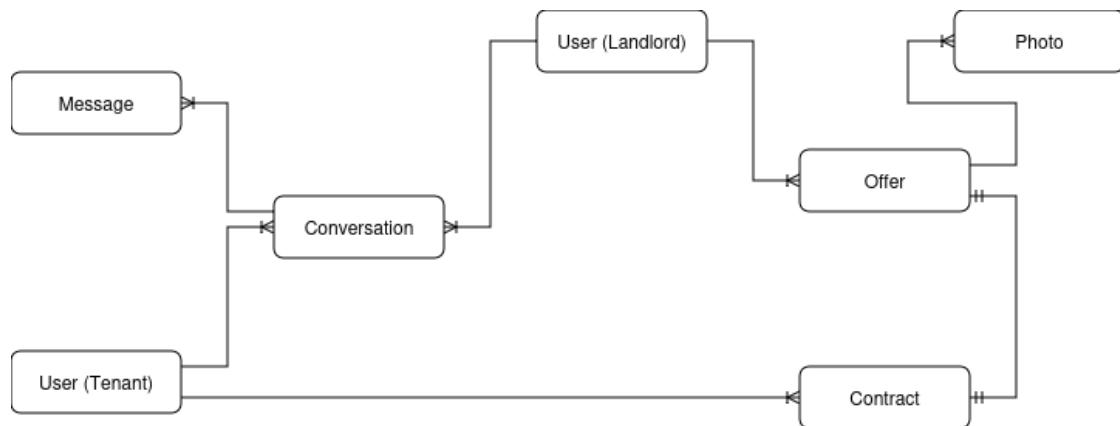
```

21 # ... i okoo 20 innych pl opisujących nieruchomości ,
22 # zgodnie z opisem w podrozdziale 5.2.3
23
24 assigned_user = models.ForeignKey(
25     TenantUser,
26     on_delete=models.SET_NULL,
27     null=True,
28     blank=True,
29     related_name='rented_offers'
30 )
31 created_at = models.DateTimeField(auto_now_add=True)
32 updated_at = models.DateTimeField(auto_now=True)
33
34 def __str__(self):
35     return self.title

```

Listing 5.1. Definicja modelu Offer w pliku models.py

Poniżej przedstawiono kluczowe modele Django, które odwzorowują struktury danych, wraz ze szczegółowym opisem każdego pola.



Rys. 5.2. Diagram ERD bazy danych.

5.2.1. Model LandlordUser

Model reprezentujący wynajmującego.

- **name:** Imię użytkownika (tekst, max 100 znaków).
- **surname:** Nazwisko użytkownika (tekst, max 100 znaków).
- **password:** Hasło użytkownika, przechowywane jako hash (tekst, max 255 znaków).

- `email`: Adres email, unikalny dla każdego użytkownika.
- `created_at`: Data i czas utworzenia konta.
- `phone_number`: Numer telefonu (opcjonalnie).
- `pesel`: Numer PESEL (opcjonalnie).
- `id_card_number`: Numer dowodu osobistego (opcjonalnie).
- `address_city`, `address_street`, `address_zip_code`: Pola adresowe (opcjonalnie).
- `bank_account_number`: Numer konta bankowego (opcjonalnie).

5.2.2. Model TenantUser

Model reprezentujący najemcę, o strukturze analogicznej do `LandlordUser`.

5.2.3. Model Offer

Model reprezentujący ofertę wynajmu.

- `title`: Tytuł oferty (tekst, max 255 znaków).
- `body`: Opis oferty.
- `user`: Klucz obcy do modelu `LandlordUser`, wskazujący na właściciela oferty.
- `status`: Status oferty (np. 'Dostępna', 'Zarezerwowana').
- `sale_or_rent`: Określa czy oferta dotyczy sprzedaży czy wynajmu.
- `location`: Lokalizacja nieruchomości.
- `price`: Cena.
- `area`: Powierzchnia w m².
- `floor`: Piętro.
- `furnished`: Poziom umeblowania.
- `number_of_rooms`: Liczba pokoi.
- `year_built`: Rok budowy.
- `condition`: Stan nieruchomości.
- `basement`: Czy do nieruchomości przynależy piwnica.
- `balcony_terrace_garden`: Informacja o przestrzeni na zewnątrz.
- `elevator`: Czy w budynku jest winda.

- **heating**: Rodzaj ogrzewania.
- **admin_fees**: Czysz adminstracyjny.
- **internet_fiber**: Dostępność internetu światłowodowego.
- **deposit**: Kaucja.
- **minimum_rental_period**: Minimalny okres wynajmu.
- **pets_allowed**: Czy zwierzęta są akceptowane.
- **additional_utilities**: Dodatkowe opłaty.
- **parking**: Dostępność parkingu.
- **building_type**: Typ budynku.
- **assigned_user**: Klucz obcy do modelu TenantUser, wskazujący na najemcę, który wynajął ofertę.

5.2.4. Model Photo

Model do przechowywania zdjęć przypisanych do oferty.

- **photo**: Pole plikowe przechowujące obraz.
- **offer**: Klucz obcy do modelu Offer.

5.2.5. Model Conversation i Message

Modele odpowiedzialne za system komunikacji.

- **Conversation**: Łączy wynajmującego, najemcę i ofertę.
- **Message**: Przechowuje treść wiadomości, informację o nadawcy, dacie wysłania oraz statusie odczytu.

5.2.6. Model Favorite

Model do przechowywania ulubionych ofert danego najemcy.

- **tenant**: Klucz obcy do modelu TenantUser.
- **offer**: Klucz obcy do modelu Offer.

5.2.7. Modele ContractTemplate i Contract

Modele do zarządzania umowami.

- **ContractTemplate**: Szablon umowy tworzony przez wynajmującego.
- **Contract**: Konkretna, wygenerowana umowa, powiązana z ofertą, najemcą i wynajmującym.

5.3. Widoki i logika biznesowa

W tej sekcji opisano działanie kluczowych widoków aplikacji. Zanim jednak żądanie HTTP dotrze do widoku, musi zostać odpowiednio skierowane przez mechanizm routingu Django. Konfiguracja ta znajduje się w plikach `urls.py`. Na Listingu 5.2 przedstawiono fragment głównego pliku konfiguracyjnego, który dodaje ścieżki z dedykowanej aplikacji.

```
1 from django.contrib import admin
2 from django.urls import path, include
3 from django.conf import settings
4 from django.conf.urls.static import static
5
6 urlpatterns = [
7     path('admin/', admin.site.urls),
8     # Doczenie cieek z aplikacji RentEaseApp
9     path('', include('RentEaseApp.urls')),
10 ]
11
12 # Serwowanie plików multimedialnych w trybie deweloperskim
13 if settings.DEBUG:
14     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Listing 5.2. Fragment głównej konfiguracji URL (`urls.py`)

Każdy widok jest funkcją lub klasą w pliku `views.py`, która przyjmuje żądanie HTTP i zwraca odpowiedź HTTP.

5.3.1. Widoki uwierzytelniania

- `loginPanel(request)`: Obsługuje logowanie użytkowników. Weryfikuje dane z formularza `LoginForm`, sprawdza poprawność hasła i w przypadku sukcesu zapisuje identyfikator użytkownika i jego typ w sesji.
- `registerPanel(request)`: Obsługuje rejestrację nowych użytkowników. Waliduje dane z formularza `RegisterForm` i tworzy nowy obiekt `LandlordUser` lub `TenantUser`.
- `logout(request)`: Wylogowuje użytkownika poprzez usunięcie danych z sesji.

5.3.2. Widoki ofert

- `mainSite(request)`: Główny widok aplikacji. Wyświetla listę wszystkich ofert, obsługuje filtrowanie i wyszukiwanie.
- `offer_detail(request, offer_id)`: Wyświetla szczegóły pojedynczej oferty.
- `my_offers(request)`: Widok dostępny tylko dla zalogowanych wynajmujących. Wyświetla listę ofert dodanych przez danego użytkownika.
- `create_offer(request)`: Umożliwia wynajmującym dodawanie nowych ofert.
- `edit_offer(request, offer_id)`: Pozwala na edycję istniejącej oferty.
- `delete_offer(request, offer_id)`: Umożliwia usunięcie oferty.

5.3.3. Widoki profili i ulubionych

- `profile(request)`: Wyświetla profil zalogowanego użytkownika i umożliwia jego edycję.
- `favorites_page(request)`: Wyświetla listę ulubionych ofert najemcy.
- `toggle_favorite(request, offer_id)`: Dodaje lub usuwa ofertę z ulubionych (działa jako endpoint API).

5.3.4. Szczegóły implementacji systemu komunikacji (Widoki konwersacji)

Wewnętrzny komunikator stanowi rdzeń interakcji między użytkownikami. Mechanika jego działania opiera się na relacji między modelami `Conversation` oraz `Message`. Aby zapewnić integralność danych i bezpieczeństwo, każdy widok przetwarzający zapytania komunikatora weryfikuje, czy użytkownik jest stroną danej konwersacji.

Na Listingu 5.3 zaprezentowano implementację metody przyjmującej zapytania typu POST, służącej do wysyłania nowych wiadomości.

```
1 def conversation_detail(request, conversation_id):  
2     # Pobranie obiektu konwersacji lub bd 404  
3     conversation = get_object_or_404(Conversation, id=conversation_id)  
4     landlord = get_logged_in_landlord(request)  
5     tenant = get_logged_in_tenant(request)  
6  
7     # Weryfikacja, czy zalogowany użytkownik jest stroną tej konwersacji  
8     if landlord and conversation.landlord == landlord:  
9         current_user, user_type = landlord, 'landlord'  
10    elif tenant and conversation.tenant == tenant:
```

```

11     current_user, user_type = tenant, 'tenant'
12
13     else:
14
15         messages.error(request, "Brak dostpu do tej konwersacji.")
16
17     return redirect('login')
18
19
20
21
22
23
24
25
26
27
28
29

```

```
# Obsuga wysymania wiadomoci (dla metody POST)
if request.method == 'POST':
    content = request.POST.get('content', '').strip()
    if content:
        Message.objects.create(
            conversation=conversation,
            sender_id=current_user.id,
            sender_type=user_type,
            content=content
        )
# Wzorzec Post/Redirect/Get
return redirect('conversation_detail', conversation_id=conversation_id)

# ... dalsza logika (oznaczanie wiadomoci jako przeczytane, renderowanie
szablonu)
```

Listing 5.3. Widok obsługujący konwersacje i wysyłanie wiadomości

W powyższym kodzie należy zwrócić uwagę na zastosowanie wzorca *Post/Redirect/Get* (PRG). Po pomyślnym zapisaniu wiadomości w bazie danych (linia 19), użytkownik jest przekierowywany (redirect) na ten sam widok za pomocą metody GET. Zapobiega to przypadkowemu, ponownemu wysłaniu wiadomości w przypadku odświeżenia strony przez użytkownika (F5).

5.3.5. Implementacja generatora umów PDF (Widoki umów)

Jedną z najbardziej rozbudowanych i innowacyjnych funkcji systemu jest generator umów najmu. Proces ten zaprojektowano tak, aby zminimalizować konieczność ręcznego wprowadzania danych. Aplikacja pobiera dane bezpośrednio ze skonfigurowanego profilu najemcy i wynajmującego, mapuje je na specjalne zmienne, a następnie konwertuje dynamiczny szablon HTML do spłaszczonego pliku PDF.

Wykorzystano do tego celu bibliotekę renderującą po stronie serwera. Przebieg tego procesu i mapowanie danych (Context) przedstawiono na Listingu 5.4.

```
1 from io import BytesIO
```

```

2 from django.http import HttpResponseRedirect
3 from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
4 from reportlab.lib.styles import getSampleStyleSheet
5 from .models import Contract
6
7 def download_contract_pdf(request, contract_id):
8     # Weryfikacja uprawnienia i pobranie obiektu umowy
9     contract = get_object_or_404(Contract, id=contract_id)
10
11    # Regeneracja treści umowy na podstawie szablonu i aktualnych danych
12    contract_content = generate_contract_from_template(
13        contract.template, contract.offer, contract.tenant, contract.landlord
14    )
15
16    # Utworzenie bufora w pamięci dla pliku PDF
17    buffer = BytesIO()
18    doc = SimpleDocTemplate(buffer, pagesize=A4)
19
20    # Lista "Flowables" - elementów do umieszczenia w PDF
21    elements = []
22    styles = getSampleStyleSheet()
23
24    # Dodanie tytułu i treści umowy
25    elements.append(Paragraph("Umowa Najmu", styles['h1']))
26    elements.append(Spacer(1, 0.2*inch))
27
28    # Podział treści na paragrafy i dodanie do dokumentu
29    for para in contract_content.split('\n'):
30        elements.append(Paragraph(para, styles['Normal']))
31        elements.append(Spacer(1, 0.1*inch))
32
33    # Zbudowanie dokumentu PDF
34    doc.build(elements)
35
36    pdf = buffer.getvalue()
37    buffer.close()
38
39    # Zwrócenie odpowiedzi HTTP z plikiem PDF
40    response = HttpResponseRedirect(pdf, content_type='application/pdf')

```

```

41     response['Content-Disposition'] = f'attachment; filename="umowa_{contract.id}.pdf"'
42
43     return response

```

Listing 5.4. Logika generowania dokumentu PDF w Django z użyciem ReportLab

Kluczowym elementem tej architektury jest całkowite odseparowanie warstwy prezentacji dokumentu (szablon `pdf_template.html`) od logiki wyciągającej dane (kontroler zdefiniowany na listingu). Dzięki temu, modyfikacja szaty graficznej samej umowy nie wymaga ponownej komplikacji ani modyfikacji kodu backendowego.

5.4. Formularze

Aplikacja wykorzystuje formularze Django do walidacji i obsługi danych wejściowych od użytkowników. W pliku `forms.py` zdefiniowano kilka kluczowych klas formularzy. Na Listingu 5.5 zaprezentowano implementację formularza modelowego ‘OfferForm’, który odpowiada za tworzenie i edycję ofert.

```

1 from django import forms
2 from .models import Offer, Photo
3
4 class OfferForm(forms.ModelForm):
5
6     photos = forms.FileField(
7         widget=forms.ClearableFileInput(attrs={'multiple': True}),
8         required=False,
9         label='Zdjęcia oferty')
10
11     class Meta:
12         model = Offer
13         exclude = ['user', 'status', 'assigned_user'] # Pola zarządzane przez logikę
14         aplikacji
15
16     def __init__(self, *args, **kwargs):
17         # Inicjalizator do dodania klas CSS do pl. formularza
18         super().__init__(*args, **kwargs)
19         for field_name, field in self.fields.items():
20             field.widget.attrs.update({'class': 'form-control'})
21
22     def save_photos(self, offer):

```

```
22     # Metoda do zapisu wielu zdj jednoczenie
23     for file in self.files.getlist('photos'):
24         Photo.objects.create(offer=offer, photo=file)
```

Listing 5.5. Implementacja formularza OfferForm w pliku forms.py

W pliku `forms.py` zdefiniowano również kilka innych kluczowych klas formularzy.

5.4.1. Formularz RegisterForm

Formularz służący do rejestracji nowych użytkowników.

- `user_type`: Pole wyboru (landlord/tenant).
- `name, surname, email, password`: Podstawowe dane użytkownika.
- `save()`: Metoda haszuje hasło i tworzy odpowiedni typ użytkownika.

5.4.2. Formularz LoginForm

Prosty formularz do logowania, zawierający pola `email` i `password`.

5.4.3. Formularz ProfileForm

Formularz do edycji profilu użytkownika. Oprócz podstawowych danych, zawiera pola na informacje kontaktowe i dane wymagane do umów.

- `clean_email()`: Metoda walidująca unikalność adresu email.
- `clean_pesel()`: Metoda sprawdzająca poprawność numeru PESEL.
- `save()`: Metoda aktualizująca dane użytkownika.

5.4.4. Formularz OfferForm

Rozbudowany formularz modelowy do tworzenia i edycji ofert. Jest on oparty o model `Offer` i zawiera pola odpowiadające atrybutom oferty. Dodatkowo, zawiera pole `photos` do przesyłania wielu zdjęć.

5.4.5. Formularz ContractPlaceholderForm

Specjalny formularz, który nie jest przeznaczony do zapisu danych, lecz do definiowania dostępnych zmiennych w szablonach umów. Metoda `get_field_groups()` zwraca pogrupowane pola, co ułatwia ich prezentację w interfejsie użytkownika.

5.5. Zarządzanie plikami statycznymi

Pliki statyczne (CSS, JavaScript, obrazy) są zarządzane zgodnie z konwencją Django. Każda aplikacja może posiadać własny katalog `static`, w którym przechowuje swoje zasoby. W środowisku deweloperskim Django automatycznie serwuje te pliki. W środowisku produkcyjnym, za pomocą polecenia `collectstatic`, wszystkie pliki statyczne są zbierane do jednego, centralnego katalogu, z którego następnie serwuje je serwer Nginx.

5.6. API

Aplikacja udostępnia prosty interfejs REST API do obsługi niektórych funkcjonalności wykonywanych asynchronicznie, takich jak dodawanie ofert do ulubionych. Na Listingu 5.6 pokazano implementację widoku, który obsługuje ten endpoint.

```
1 from django.http import JsonResponse
2 from django.views.decorators.http import require_POST
3 from .models import Offer, Favorite
4 from .auth import get_logged_in_tenant
5
6 @require_POST # Widok akceptuje tylko metod POST
7 def toggle_favorite(request, offer_id):
8     tenant = get_logged_in_tenant(request)
9     if not tenant:
10         return JsonResponse({'status': 'error', 'message': 'Brak uwierzytelnienia'}, status=401)
11
12     try:
13         offer = Offer.objects.get(id=offer_id)
14         # Sprawdzenie, czy obiekt już istnieje i jego usunięcie lub utworzenie
15         fav, created = Favorite.objects.get_or_create(tenant=tenant, offer=offer)
16
17         if created:
18             action = 'added'
19         else:
20             fav.delete()
21             action = 'removed'
22
23     return JsonResponse({'status': 'ok', 'action': action})
```

```
24  
25     except Offer.DoesNotExist:  
26         return JsonResponse({'status': 'error', 'message': 'Oferta nie istnieje'},  
status=404)
```

Listing 5.6. Widok API do zarządzania ulubionymi (views.py)

Wykorzystanie API i zapytań JavaScript po stronie klienta pozwala na dynamiczną zmianę interfejsu (np. zmianę ikony serca) bez konieczności przeładowywania całej strony.

5.7. Bezpieczeństwo systemu

Podczas tworzenia systemów webowych przetwarzających dane osobowe (w tym numery PESEL i dane z dowodów osobistych), kwestie cyberbezpieczeństwa stanowią priorytet. W projekcie RentEase zaimplementowano wielowarstwowe mechanizmy obronne:

- **Zabezpieczenie przed Cross-Site Request Forgery (CSRF):** Django natywnie chroni aplikację przed atakami polegającymi na wymuszaniu fałszywych żądań. W każdym formularzu przesyłającym dane dodano znacznik `{% csrf_token %}`, który generuje unikalny kryptograficznie ciąg znaków, weryfikowany następnie przez serwer przy użyciu Middleware.
- **Ochrona przed SQL Injection:** Dzięki użyciu systemu mapowania ORM, programista nie pisze surowych zapytań języka SQL. Django automatycznie sanityzuje (odkaża) i binduje zmienne przekazywane w żądaniach HTTP, zanim trafią one do silnika bazy danych PostgreSQL.
- **Zarządzanie sekretami (Environment Variables):** Klucz kryptograficzny aplikacji (`SECRET_KEY`), poświadczania do bazy danych oraz ustawienia trybu `DEBUG` nie są na stałe zapisane w kodzie źródłowym. Są one pobierane w czasie działania aplikacji z pliku konfiguracyjnego `.env`, który z kolei jest wykluczony ze śledzenia systemu kontroli wersji Git (plik `.gitignore`).

5.8. Testowanie oprogramowania (Unit Testing)

Ostatnim, ale równie ważnym elementem inżynierii oprogramowania jest testowanie. W projekcie RentEase zaimplementowano zautomatyzowane testy jednostkowe (ang. *Unit*

Tests). Wykorzystano w tym celu standardową bibliotekę języka Python `unittest` zintegrowaną z modelem testowym frameworka Django.

Testy skupiają się na sprawdzeniu poprawności modeli danych, np. procesu prawidłowej rejestracji ról użytkownika, a także gwarantują poprawność procesu szyfrowania hasła przed zapisem do bazy. Poniżej na Listingu 5.7 zaprezentowano przykład testu weryfikującego te założenia.

```
1 from django.test import TestCase
2 from .models import LandlordUser, TenantUser
3 from django.contrib.auth.hashers import make_password, check_password
4
5 class UserModelTest(TestCase):
6     def setUp(self):
7         # Kod wykonywany przed każdym testem
8         password_raw = 'TestPassword123!'
9         self.password_hashed = make_password(password_raw)
10        self.tenant = TenantUser.objects.create(
11            email='anna.najemca@example.com',
12            password=self.password_hashed,
13            name='Anna',
14            surname='Najemca'
15        )
16
17    def test_tenant_creation_and_fields(self):
18        """Sprawdzenie poprawnosci zmapowanych atrybutow."""
19        self.assertEqual(self.tenant.email, 'anna.najemca@example.com')
20        self.assertEqual(self.tenant.name, 'Anna')
21
22    def test_password_hashing(self):
23        """Weryfikacja szyfrowania hasla uzytkownika."""
24        self.assertNotEqual(self.tenant.password, 'TestPassword123!')
25        self.assertTrue(check_password('TestPassword123!', self.tenant.password))
```

Listing 5.7. Przykład testu jednostkowego walidującego model użytkownika

Napisanie scenariuszy testowych w tej formie umożliwia ich błyskawiczne i cykliczne wykonywanie. Dzięki temu, w przypadku wprowadzania nowych funkcjonalności w przyszłości, programista od razu dowie się, czy jego najnowszy kod nie uszkodził istniejących już, podstawowych mechanizmów działania platformy (tzw. testy regresyjne).

5.9. Niestandardowe tagi szablonów

System szablonów Django jest wysoce elastyczny i pozwala na tworzenie własnych tagów i filtrów, które upraszczają logikę w warstwie prezentacji. W projekcie zdefiniowano niestandardowy filtr do formatowania wartości pieniężnych, aby zapewnić spójne i czytelne wyświetlanie cen w całej aplikacji. Implementację takiego filtra przedstawiono na Listingu 5.8.

```
1 from django import template
2 from django.utils.numberformat import format
3
4 register = template.Library()
5
6 @register.filter(name='currency')
7 def currency(value):
8     """
9     Filtre do formatowania liczby jako waluty (PLN).
10    Przykaz uycia w szablonie: {{ offer.price|currency }}
11    """
12
13    try:
14        # Formatowanie liczby do dwch miejsc po przecinku,
15        # z separatorem tysicy (spacja) i przecinkiem dziesitnym.
16        formatted_value = format(value, decimal_sep=',', thousand_sep=' ', grouping
17        =3)
18        return f'{formatted_value} z'
19    except (ValueError, TypeError):
20        return value
```

Listing 5.8. Niestandardowy filtr szablonu do formatowania waluty

Dzięki takiemu rozwiązaniu, cała logika formatowania jest zamknięta w jednym miejscu, a szablony pozostają czyste i czytelne. Zamiast pisać skomplikowaną logikę w widoku lub szablonie, wystarczy proste wywołanie ‘offer.price|currency’.

5.10. Niestandardowe polecenia zarządzania

Django pozwala na tworzenie własnych poleceń (komend) uruchamianych z poziomu wiersza poleceń za pomocą skryptu ‘manage.py’. Jest to niezwykle użyteczne do automatyzacji zadań administracyjnych, takich jak czyszczenie bazy danych, import danych

czy generowanie raportów. W projekcie RentEase zaimplementowano polecenie służące do usuwania starych, nieaktywnych ofert, co przedstawia Listing 5.9.

```
1 from django.core.management.base import BaseCommand
2 from django.utils import timezone
3 from datetime import timedelta
4 from ..models import Offer
5
6 class Command(BaseCommand):
7     help = 'Usuwa oferty, ktre s nieaktywne (status "Wynajta") od ponad 90 dni.'
8
9     def handle(self, *args, **options):
10         ninety_days_ago = timezone.now() - timedelta(days=90)
11
12         # Znalezienie starych, wynajtych ofert
13         old_offers = Offer.objects.filter(
14             status='Rented',
15             updated_at__lte=ninety_days_ago
16         )
17
18         count = old_offers.count()
19
20         if count > 0:
21             old_offers.delete()
22             self.stdout.write(self.style.SUCCESS(f'Pomylnie usunuto {count} starych
23             ofert.'))
24         else:
25             self.stdout.write(self.style.NOTICE('Nie znaleziono ofert do usunienia.'))
26
```

Listing 5.9. Polecenie zarządzania do usuwania starych ofert

Takie polecenie może być następnie uruchamiane ręcznie ('python manage.py cleanupoffers') lub cyklicznie zapomocą systemowegonarzędzia, takiego jak cron, copozwalanapełn

6. Podsumowanie i wnioski

Celem niniejszej pracy inżynierskiej było zaprojektowanie, zaimplementowanie oraz wdrożenie kompletnej aplikacji webowej RentEase, przeznaczonej do kompleksowego zarządzania procesem najmu nieruchomości. Głównym założeniem było stworzenie platformy, która nie tylko agreguje oferty, ale również dostarcza narzędzi usprawniających komunikację i finalizację transakcji, odpowiadając tym samym na zidentyfikowane w analizie biznesowej problemy rynku najmu w Polsce.

Wszystkie założone cele projektu zostały pomyślnie zrealizowane. Zbudowano w pełni funkcjonalny system w oparciu o framework Django i język Python, który spełnia zdefiniowane wymagania funkcjonalne i niefunkcjonalne. Użytkownicytrzymali możliwość rejestracji kont w podziale na role (Wynajmujący i Najemca), zarządzania ofertami, bezpiecznej komunikacji oraz, co stanowi kluczową innowację, generowania spersonalizowanych umów najmu w formacie PDF. Interfejs użytkownika został zaprojektowany z myślą o responsywności, zapewniając komfortowe użytkowanie na różnych urządzeniach.

Największym wyzwaniem technicznym w trakcie realizacji projektu była implementacja modułu do generowania umów w taki sposób, aby był on zarówno elastyczny dla wynajmującego, jak i prosty w obsłudze. Wykorzystanie biblioteki ReportLab pozwoliło na dynamiczne tworzenie dokumentów PDF po stronie serwera, z uwzględnieniem polskich znaków diakrytycznych, co było kluczowe dla zachowania formalnego charakteru umów. Innym istotnym aspektem była implementacja niestandardowego systemu uwierzytelniania, który musiał obsługiwać dwa odrębne typy użytkowników z różnymi danymi profilowymi, przy jednoczesnym zachowaniu wysokiego poziomu bezpieczeństwa.

Zrealizowany projekt z powodzeniem dowodzi, że możliwe jest stworzenie scenaryjnej platformy, która integruje cały cykl życia procesu najmu – od publikacji oferty po podpisanie umowy. Aplikacja RentEase stanowi solidną podstawę, którą można dalej rozwijać.

6.1. Kierunki dalszego rozwoju

Stworzona aplikacja posiada znaczący potencjał rozwojowy. Wśród możliwych do zaimplementowania w przyszłości funkcjonalności znajdują się:

- **Integracja z systemami płatności online:** Umożliwienie automatyzacji procesu opłacania czynszu oraz kaucji za pośrednictwem operatorów płatności, takich jak Stripe, PayU czy Przelewy24. Funkcjonalność ta znacząco podniosłaby wygodę i bezpieczeństwo transakcji.
- **Zaawansowana weryfikacja tożsamości:** Integracja z usługami weryfikacji tożsamości (np. mojeID), co pozwoliłoby na zwiększenie zaufania między stronami i ograniczenie ryzyka oszustw.
- **Aplikacje mobilne:** Stworzenie dedykowanych aplikacji mobilnych dla systemów Android i iOS, które umożliwiłyby jeszcze wygodniejszy dostęp do platformy i wykorzystanie powiadomień push do informowania o nowych wiadomościach czy statusie umowy.
- **Moduł dla agencji nieruchomości:** Wprowadzenie specjalnego typu konta dla agencji, pozwalającego na zarządzanie portfelem wielu nieruchomości oraz kontami poszczególnych agentów.
- **System ocen i reputacji:** Umożliwienie najemcom i wynajmującym wzajemnego oceniania się po zakończonej transakcji, co pomogłoby w budowaniu wiarygodności użytkowników platformy.
- **Wykorzystanie uczenia maszynowego:** Implementacja mechanizmów sztucznej inteligencji do sugerowania optymalnej ceny najmu na podstawie parametrów nieruchomości i danych rynkowych, a także do lepszego dopasowywania ofert do preferencji najemców.

Podsumowując, projekt RentEase zakończył się sukcesem, a stworzona aplikacja stanowi w pełni funkcjonalny produkt o realnej wartości użytkowej. Proces jej tworzenia pozwolił na praktyczne zastosowanie wiedzy z zakresu inżynierii oprogramowania i technologii webowych, a potencjalne kierunki dalszego rozwoju otwierają szerokie możliwości na przyszłość.

Literatura

- [1] Oficjalna strona projektu Django: <https://www.djangoproject.com/>
- [2] Oficjalna strona biblioteki Pillow: <https://pypi.org/project/pillow/>
- [3] Oficjalna strona projektu L^AT_EX: <https://www.latex-project.org/>
- [4] Narzędzie do modelowania systemów diagrams.net (draw.io):
<https://app.diagrams.net/>
- [5] Model sztucznej inteligencji Google Gemini, wykorzystany do wygenerowania logo-typu aplikacji: <https://gemini.google.com/>
- [6] Otodom, "Podsumowanie roku na rynku najmu", 2023. Dostępne online:
<https://media.otodom.pl/informacje-prasowe/podsumowanie-roku-na-rynku-najmu>
- [7] Oficjalna strona projektu Bootstrap: <https://getbootstrap.com/>

STRESZCZENIE PRACY DYPLOMOWEJ INŻYNIERSKIEJ

APLIKACJA WEBOWA DO ZARZĄDZANIA PROCESEM NAJMU NIERUCHOMOŚCI Z OBSŁUGĄ DOKUMENTACJI I POWIADOMIEŃ

Autor: Szymon Matyka, nr albumu: EF-173668

Opiekun: dr inż. Grzegorz Drałus

Słowa kluczowe: (Django, Python, zarządzanie najmem, aplikacja webowa, generowanie umów)

Celem pracy było zaprojektowanie i zaimplementowanie aplikacji webowej RentEase, służącej do kompleksowego zarządzania procesem najmu nieruchomości. Aplikacja, zbudowana w oparciu o framework Django, usprawnia komunikację między wynajmującymi a najemcami, automatyzuje procesy związane z ofertami oraz wprowadza innowacyjne rozwiązanie w postaci zintegrowanego generatora umów najmu. W pracy opisano cały proces tworzenia aplikacji, od analizy biznesowej, przez projektowanie architektury i implementację, aż po wdrożenie. Przedstawiono również szczegółową dokumentację techniczną zrealizowanego projektu.

BSC THESIS ABSTRACT

A WEB APPLICATION FOR MANAGING THE PROPERTY RENTAL PROCESS, INCLUDING DOCUMENTATION AND NOTIFICATIONS.

Author: Szymon Matyka, Student's ID: EF-173668

Supervisor: Dr Eng. Grzegorz Drałus

Key words: (Django, Python, rental management, web application, contract generation)

The aim of this thesis was to design and implement the RentEase web application for comprehensive management of the real estate rental process. The application, built on the Django framework, streamlines communication between landlords and tenants, automates offer-related processes, and introduces an innovative solution in the form of an integrated lease agreement generator. The thesis describes the entire application development process, from business analysis, through architecture design and implementation, to deployment. Detailed technical documentation of the completed project is also presented.