POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Zuzanna Matejczyk, 11241

Grzegorz Gaweł, 9578

Paweł Budzowski, 9691

Sebastian Miodek, 9522

Paweł Hnatyk, 8259

Maciej Pawlina, 7972

Rafał Drozd, 8624

Kamil Bednarz, 9433

System zarządzania wspólnotami mieszkaniowymi

Praca inżynierska napisana pod kierunkiem:

dr inż. Mariusza Trzaski

Warszawa, czerwiec 2015

Streszczenie

Celem projektu jest stworzenie narzędzia wspierającego działalność wspólnot mieszkaniowych. System ma wspomóc procesy zarządzania, poprawić komunikację zarząd – mieszkaniec – administrator oraz ułatwić wypełnianie obowiązków zarówno pracowników, jak i mieszkańców. Jest to medium społecznościowe, którego głównym zadaniem jest nie tylko ułatwienie zarządzania wspólnotą, ale także komunikacja pomiędzy mieszkańcami, dzięki możliwości dodawania ogłoszeń i pomysłów na inwestycje, które urozmaicą i wzbogacą wspólne miejsce zamieszkania.

Spis treści

Wstęp

Mamy obecnie do czynienia z szybkim tempem rozwoju zabudowy mieszkalnej wielorodzinnej, zwłaszcza w dużych miastach. Dominującą formą zarządzania, jaka temu rozwojowi towarzyszy, jest wspólnota mieszkaniowa.

1.1. Cel i zakres pracy

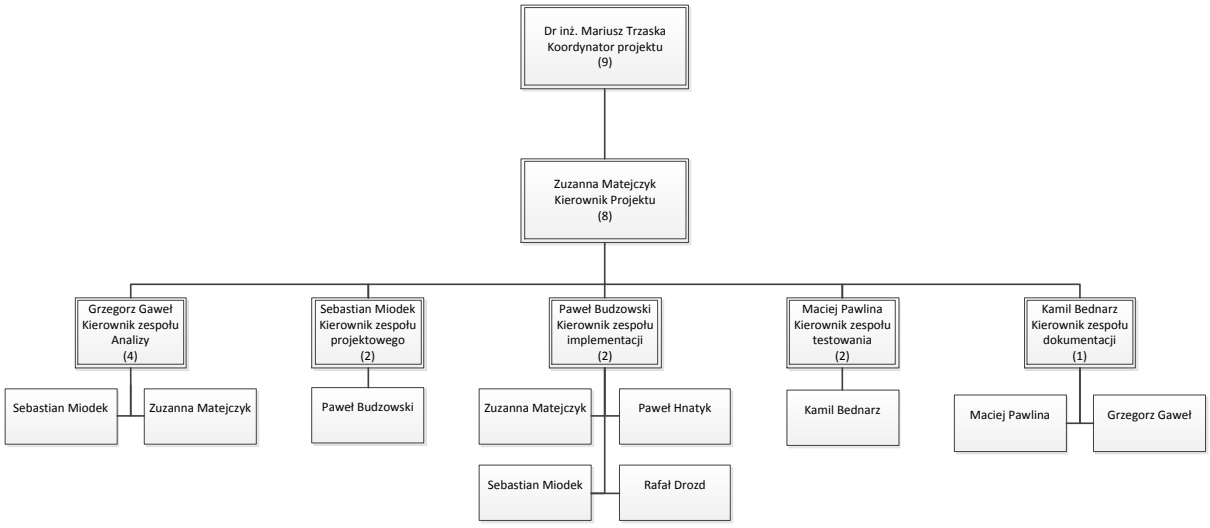
Celem niniejszej pracy inżynierskiej jest stworzenie aplikacji wspierającej zarządzanie wspólnotami mieszkaniowymi. Projekt został zrealizowany w ramach pracy inżynierskiej, realizowanej na specjalizacji Inżynieria Oprogramowania i Baz Danych w Polsko-Japońskiej Akademii Technik Komputerowych.

1.2. Organizacja pracy

Praca inżynierska jest podsumowaniem pracy całego zespołu. Dokumentacja ta jest prezentacją powstałej aplikacji, opisem produktów konkurencyjnych, a także zastosowanych technologii oraz funkcjonalności. Dokument zakończony jest słownikiem pojęć, wykazem tabel i rysunków oraz spisem literatury.

1.3. Struktura organizacyjna zespołu

Rysunek 1 przedstawia strukturę organizacyjną zespołu projektowego wraz z przydzielonymi funkcjami.



Rysunek 1. Struktura organizacyjna zespołu. ***Źródło: Opracowanie własne.***

1.3.1 Kierownik Projektu

Osoba odpowiedzialna za planowanie i realizację projektu, jego jakość i zgodność z wymaganiami klienta. Do jego zadań należy określenie wymagań projektu, zarządzanie czasem, kosztem i zasobami ludzkimi. Jego rolą jest przydzielenie pracowników do odpowiednich grup i

kontrola nad postępem prac, które wcześniej im przydzielił. Kierownik Projektu powołuje kierowników zespołów.

1.3.2 Kierownik Zespołu

Jego zadaniem jest kierowanie konkretnym zespołem, przydzielanie zadań w zespole oraz określanie czasu na ich wykonanie. Jest odpowiedzialny za osiągnięcia całego zespołu, motywuje uczestników projektu oraz dostosowuje zadania do ich umiejętności.

1.3.3 Zespół Projektowy

Tworzy strategie budowania systemu, m.in. platformy sprzętowej i systemu operacyjnego, technologii przechowywania danych oraz technologii wykonania. Jest odpowiedzialny za transformację wymagań do postaci projektowej, stworzenie GUI oraz projektu bazy danych.

1.3.4 Zespół Implementacji

Odpowiedzialny jest za stworzenie kodu źródłowego aplikacji zgodnie z wymaganiami zespołu analizy i rozwiązaniami zespołu projektowego.

1.3.5 Zespół Testowania

Testowanie aplikacji zgodnie z przypadkami użycia, wykrywanie błędów i ich zgłaszanie, sprawdzanie, czy oprogramowanie jest zgodne ze specyfikacją i wymaganiami użytkownika. Głównym celem tego zespołu jest wykrycie jak największej liczby usterek i poprawienie ich przed dostarczeniem aplikacji klientowi.

1.3.6 Zespół Dokumentacji

Zadaniem zespołu jest napisanie dokumentacji we współpracy ze wszystkimi pozostałymi zespołami. Członkowie tego zespołu są odpowiedzialni za zbieranie materiałów oraz korektę językową pracy.

1.3.7 Zespół Analizy

Dąży do stworzenia modelu systemu opartego na spisie wymagań klienta – poprawnego, kompletnego, spójnego i wolnego od niejednoznaczności. Dokonuje transformacji przypadków użycia zdefiniowanych podczas zbierania wymagań w model obiektowy, który będzie kompletnym opisem systemu. Zespół jest odpowiedzialny za to, aby wszystkie niejasności i niespójności zostały wyjaśnione.

1.3.8 Uczestnik Projektu

Jego celem jest realizacja zadań powierzonych przez kierownika zespołu. Zgłasza wszystkie nieprawidłowości i problemy napotkane po drodze, które mogą doprowadzić do opóźnienia wykonania zadania lub projektu.

1.4. Zakres prac

Tabela 1 przedstawia zakres zadań wykonanych przez poszczególne osoby, ich przynależność do zespołów oraz dodatkowe role jakie pełnili w projekcie.

Tabela 1. Zakres prac wykonanych przez poszczególnych uczestników. Źródło: Opracowanie własne.

Osoba	Zespoły	Dodatkowe role w zespole	Wykonane zadania
Maciej Pawlina	Testy, Dokumentacja	Kierownik testów	• Stworzenie diagramu UC oraz scenariuszy dla aktora administrator systemu, • Napisanie metod zapisujących DAO dla 1/3 klas, • Napisanie klas dodających dane testowe dla 1/3 klas, • Testy funkcjonalności FAZA 1 (1/2), FAZA 2, FAZA 3 (1/2), • Dokumentacja, opis części zastosowanych technologii, • Bieżące testowanie aplikacji celem wykrycia błędów oraz ich zgłaszanie zespołowi implementacyjnemu, • Sprawdzanie czy oprogramowanie jest zgodne ze specyfikacja i wymaganiami użytkownika, • Stworzenie listy ekranów uprawnień i ich testy po zaimplementowaniu.
Paweł Hnatyk	Implementacja		• Wykonanie słownika, • Tłumaczenie diagramu klas, • Utworzenie klas modelu oraz DAO (1/2), • Utworzenie kontrolerów (1/2), • Wykonanie widoku: ustaw, moje mieszkanie, liczniki, • Implementacja kalendarza, inwestycji, wykresów w statystykach, • Implementacja filtrowania oraz jego opis do dokumentacji, • Bugfixing i zmiany funkcjonalne w trakcie .
Sebastian Miodek	Analiza, Implementacja	Kierownik projektowania	• Stworzenie wymagań funkcjonalnych, • Stworzenie diagramów sekwencji, • Stworzenie diagramu

			komponentów, • Stworzenie scenariuszy oraz diagramu UC dla aktora właściciel mieszkania, • Stworzenie scenariuszy dla aktora lokator, • Stworzenie widoku listy i dodawania płatności, • Stworzenie widoku wspólnoty, • Stworzenie widoku ogłoszeń Bootstrap, • Stworzenie widoku awarii, • Stworzenie widoku ankiet, • Implementacja statystyk oraz jej opis, • Implementacja sortowania tabel, • Bugfixing
Paweł Budzowski	Projektowanie, Implementacja	Kierownik implementacji	• Organizacja środowiska do zarządzania projektem i pomoc w jego konfiguracji reszcie zespołu, • Organizacja repozytorium kodu źródłowego oraz środowiska testowego, • Opracowanie i implementacja architektury aplikacji, • Stworzenie projektowego diagramu klas, • Implementacja trwałości, transakcyjności, mechanizmów obsługi błędów i bezpieczeństwa (w tym model uprawnień, uwierzytelnianie, szyfrowanie hasła), • Implementacja kontrolerów: Kalendarza (1/2), Zarządzania wspólnotą (1/2), Strony głównej (1/2), Płatności (1/2), Zarządzania użytkownikami, rolami i uprawnieniami, • Implementacja widoków: Logowania (1/2), Błędów, Zarządzania wspólnotą, Zarządzania użytkownikami, rolami i uprawnieniami, Zmiany hasła, Zgłoszeń inwestycji (1/2), • Implementacja filtrowania po wspólnotach i selekcji

			wspólnoty, filtrowania względem uprawnień, kontroli dostępu w kontrolerach i widokach, • Definicja makr wykorzystywanych w widokach, • Implementacja mechanizmu stronicowania (1/2), • Implementacja dialogów i powiadomień, • Poprawki w częściach kodu oddanych przez innych, • Bugfixing, • Wsparcie testów, • Opis architektury systemu oraz implementacji modelu uprawnień.
Zuzanna Matejczyk	Analiza, Implementacja	Kierownik projektu	• Organizacja środowiska do zarządzania projektem, • Stworzenie diagramu UC i scenariuszy dla aktora administrator, • Stworzenie diagramów aktywności, • Utworzenie klas modelu oraz DAO (1/2), • Utworzenie kontrolerów (1/2), • Napisanie metod zapisujących DAO dla 1/3 klas, • Napisanie klas dodających dane testowe dla 1/3 klas, • Rozszerzenie aplikacji o dodatkową encję "Operacja finansowa" (klasa modelu, DAO, kontroler, service, dodawanie testowych danych), oraz stworzenie widoku do niej, • Implementacja inwestycji i projektów wraz z automatycznym tworzeniem inwestycji z projektu po jego akceptacji, • Implementacja mechanizmu stronicowania, deklaracji liczby mieszkańców, wyliczania czynszów , • Bugfixing , • Planowanie, koordynacja i monitorowanie prac całego

			zespołu, • Korekta językowa i stylistyczna dokumentacji.
Grzegorz Gaweł	Analiza, Dokumentacja	Kierownik analizy	• Przegląd i opis problematyki, • Koordynacja prac nad diagramem klas oraz jego uzupełnienie, • Stworzenie scenariuszy dla aktora zarząd, • Konsolidacja diagramów UC tworzonych przez różne osoby w jeden, • Kontrola zgodności prac implementacyjnych z dokumentacją, • Wykonanie generowania danych do wyświetlania, • Przygotowanie treści informacyjnych na stronę WWW, • Dokumentacja, opis zastosowanych technologii (1/3) oraz aktualizacja i poprawa diagramów klas, stanów, sekwencji, UC oraz scenariuszy.
Rafał Drozd	Analiza, Implementacja		• Przegląd i opisanie dostępnych rozwiązań, • Stworzenie wymagań niefunkcjonalnych, • Poprawki wyglądu aplikacji, • Implementacja ogłoszeń na stronie głównej, • Wykonanie projektu logo oraz elementów nawigacji, • Opis implementacji widoku „Moje mieszkanie”.
Kamil Bednarz	Dokumentacja, Testy	Kierownik dokumentacji	• Napisanie metod zapisujących DAO dla 1/3 klas, • Napisanie klas dodających dane testowe dla 1/3 klas, • Zbiór materiałów przydatnych przy pisaniu tekstu pracy oraz stworzenie diagramu organizacyjnego, • Stworzenie formatek GUI, • Testy funkcjonalności Fazy 1 (1/2) i Fazy 3 (1/2), • Bieżące testowanie aplikacji i zgłaszanie wykrytych

			nieprawidłowości, • Dokumentacja, opis części zastosowanych technologii (1/3), korekta materiałów dostarczonych, • Realizacja rozdziałów: I, II, III (poprawa), IV, VIII, • Szereg prac z formatowaniem, poprawianiem, i aktualizowaniem dokumentacji do stanu faktycznego aplikacji.

2. Wprowadzenie do problematyki

Wspólnoty wypierają popularne niegdyś spółdzielnie, jednakże ze względu na swoją specyfikę – zarówno w aspekcie wpływu mieszkańców na ich decyzje, jak i realizowanie przez nich swoich praw i obowiązków – wymagają właściwej obsługi. Wspólnoty, mimo że część stosowanych w nich rozwiązań nie różni się znacząco od tych właściwych spółdzielniom mieszkaniowym, umożliwiają znacznie większy niż to miało miejsce w spółdzielniach, wpływ mieszkańców na funkcjonowanie osiedla. Każdy z członków wspólnoty uprawniony jest do wglądu we wszelkie informacje dotyczące tejże wspólnoty, posiada również prawo do uczestnictwa w podejmowaniu decyzji w stopniu zależnym od posiadanego udziału w nieruchomości wspólnej.

Wspólnota mieszkańców reprezentowana jest przez tzw. zarząd wspólnoty, wybierany przez członków wspólnoty. Zarząd reprezentuje wspólnotę w stosunkach zewnętrznych, w szczególności podpisuje w jej imieniu umowy, np. związane z gospodarką odpadami, umowy z dostarczycielem energii dla części wspólnych nieruchomości, umowy o ochronę nieruchomości, umowy z usługodawcami zapewniającymi usługi ogrodnicze czy sprzątanie obiektów, dokonuje w imieniu wspólnoty płatności, zatwierdza także poprawne wykonanie umów przez dostarczycieli usług i towarów.

Do zadań zarządu należy również dbałość o stan osiedla (budynków, elewacji, ścieżek, powierzchni zielonych itp.). Zazwyczaj zarząd nie zajmuje się tym sam, głównie z uwagi na brak koniecznych kwalifikacji, zatrudniając bądź „outsource'ując" tego typu usługi z zakresu nadzoru technicznego, budowlanego, elektrycznego itp. Niejednokrotnie usługi te są nabywane przez wspólnoty niejako razem z usługą administrowania osiedlem. Firmy administrujące nieruchomościami są bardzo popularne na polskim rynku nieruchomości, oferują zazwyczaj kompleksowe usługi – od zarządzania poprzez usługi specjalistów, np. elektryków, inspektorów budowlanych, aż do księgowości.

Zarządcy przekonują, że wspólnoty są efektywniejsze, lecz wyraźnie podkreślają - są większym obowiązkiem dla lokatorów. To lokator decyduje, na co są wydawane jego pieniądze. Potwierdzeniem efektywności jest ankieta (Rysunek 2) która przedstawia zarządzanie czynszem.

Rysunek 2. Ankieta efektywności w zarządzaniu czynszem. ***Źródło: [1].***

Często członkowie wspólnot są zbyt zajęci i lekceważą, bądź zapominają o udziale w zebraniach. Na posiedzeniach podejmowane są najważniejsze uchwały dotyczące kosztów zarządzania, remontów, konserwacji czy też inwestycji bądź opłat. Przedstawiane jest także sprawozdanie zarządu bądź ustalanie nowego składu, czyli wszystko, co ma wpływ na efektywność i sprawne funkcjonowanie wspólnoty. Wymagane jest, aby takie spotkanie odbyło się co najmniej raz w roku oraz aby o jego odbyciu, właściciele mieszkań zostali poinformowani co najmniej tydzień wcześniej. Niepoinformowanie przez zarząd o terminie spotkania właściciela może stać się podstawą do anulowania uchwał i innych ustaleń przeprowadzonych na zebraniu. Jak widzimy, zaniedbania w przepływie informacji mogą doprowadzić do zaburzeń w funkcjonowaniu wspólnoty, a z tym wiąże się brak jej rezultatów. Brak efektywności niesie za sobą koszty, a te wywołują niezadowolenie mieszkańców.

Kolejną kwestią jest bezpieczeństwo środków i mienia powierzonych przez mieszkańców do zagospodarowania przez zarząd. Zarząd może być jedno lub kilku osobowy, a jego członkami mogą być właściciele lokali, jak również osoby spoza ich grona. Niejednokrotnie w mediach słyszymy o celowych działaniach na szkodę firmy bądź innej organizacji. Gdy przeanalizujemy wyżej wymienioną zasadę funkcjonowania wspólnoty, dojdziemy do wniosku, że nie mamy kontroli nad bieżącym stanem wspólnoty. Informacje takie są przekazywane tylko podczas sprawozdań. Dlaczego więc członek nie może mieć bezpośredniej kontroli nad zarządem, jeżeli jest uprawniony do wglądu we wszystkie informacje dotyczące wspólnoty?

Doświadczenie pokazuje, że funkcjonowanie przedstawionej powyżej wspólnoty mogłoby zostać znacznie usprawnione poprzez zastosowanie odpowiednich rozwiązań informatycznych. Rozwiązania takie zarówno pozwalałyby na sprawniejsze wykonywanie zadań zarządu, jak również spełniałyby funkcję informacyjną. Ponadto umożliwiałyby mieszkańcom wpływ na funkcjonowanie osiedla, chociażby poprzez zgłaszanie zaobserwowanych na osiedlu usterek czy problemów bądź zgłaszanie pomysłów związanych z możliwymi inwestycjami. Według posiadanych przez nas informacji, nie zostało dotąd stworzone kompleksowe rozwiązanie o charakterze informatycznym, oferujące wyżej wymienione funkcjonalności w formie powiązanych ze sobą aplikacji webowych i mobilnych, pojawia się zatem pole dla stworzenia aplikacji realizowanej w ramach niniejszej pracy inżynierskiej.

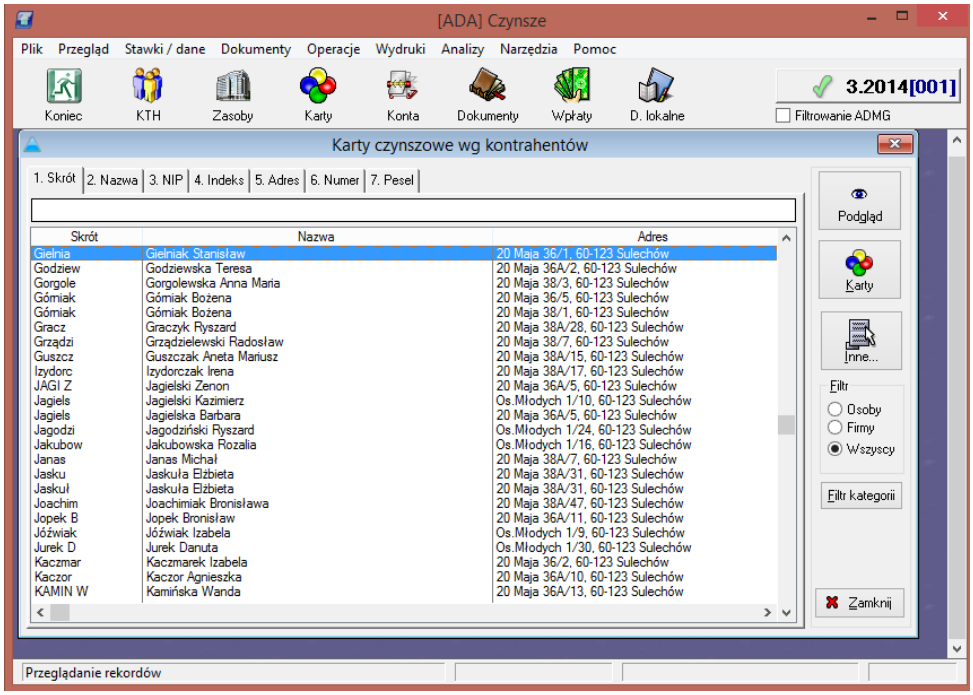
3. Analiza istniejących aplikacji do zarządzania wspólnotami mieszkaniowym

Rynek oferuje nam dość dużo aplikacji służących zarządzaniu wspólnotami. Aby dobrze poznać aplikacje konkurencyjne do naszego projektu, postanowiliśmy każdą z nich zainstalować i osobiście przetestować. Testy obejmowały dokumentację wyglądu oraz opis poszczególnych rozwiązań. Badania te pozwoliły na lepsze zapoznanie się z ich funkcjami i budową, a co najważniejsze, ujawniły ich wady i zalety, o których powiemy pod koniec rozdziału.

3.1. ADA firmy Pixel

Program ADA stworzony przez Zakład Informatyki Stosowanej Pixel. System pozwala na wykonywanie zasadniczych operacji związanych z obsługą spółdzielni mieszkaniowej. Składa się z poszczególnych modułów.

- **Czynsze (**Rysunek 3**):** Moduł pozwala na wykonanie operacji związanych z rozliczaniem czynszu za lokale i inne przestrzenie użytkowe, które wchodzą w skład wspólnoty. Dokonuje się tutaj rejestracji wpłat oraz korekt. Wszystko może zostać wprowadzone ręcznie bądź zaimportowane z modułu kasa albo rachunku wirtualnego. Podatek VAT dla wcześniej wskazanych lokali naliczany jest automatycznie według wcześniej zdefiniowanych stawek. Dzięki modułowi wystawimy także fakturę lub rachunek.



Rysunek 3. Karty czynszowe. ***Źródło: Opracowanie własne.***

- **Rachunek wirtualny**: Jest dodatkiem do modułu Czynsze, który realizuje import wpłat czynszowych na indywidualne rachunki lokatorów.
- **Rozliczenia CO**: jest dodatkiem do programu który realizuje import i eksport danych do firmy rozliczającej CO. Można go zaadaptować do realizacji innych rozliczeń.
- **Media**: Moduł daje możliwość rejestrowania liczników (wody zimnej, wody ciepłej itp.), dzięki niemu porównamy stan liczników i dokonamy rozliczenia zużycia mediów. Rozliczenia

obliczane są na podstawie liczników indywidualnych oraz zbiorczych,(jeżeli znajdują się w budynku). Program Media integruje się z programem Czynsze, pobiera z niego wartości naliczeń i wpłat już dokonanych. Program Czynsze przyjmuje nowe wyliczone normy zużycia. Przypomni o terminach legalizacji liczników oraz wyświetla wyniki. Posiada możliwość tworzenia listy dla inkasentów. Dzięki temu, że programy Media i Czynsze pracują łącznie, możemy generować korespondencję do lokatorów, w tym rozliczenia z obu programów.

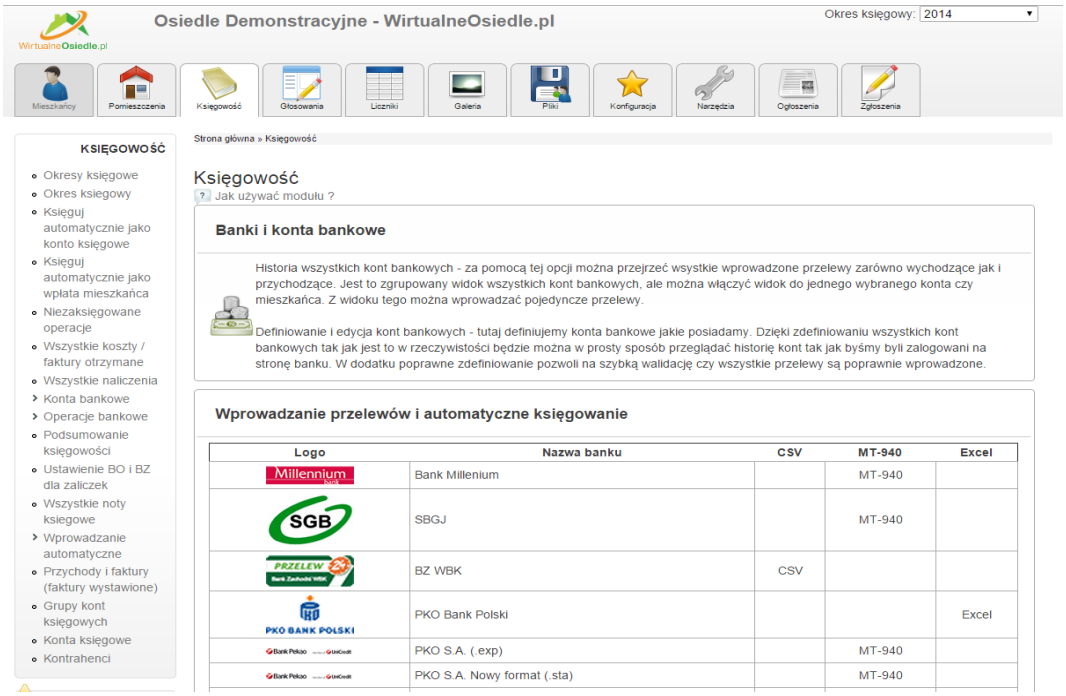
- **Finansowo-księgowy**: Stworzony do prowadzenia księgowości wspólnoty. Pozwala na założenie wieloznakowej struktury kont (40 znaków), rejestrację dokumentów księgowych – zarówno pojedynczych, jak i grupowych. Z programu wygenerujemy dowolne wykazy według wcześniej ustalonych kryteriów, napiszemy i wydrukujemy korespondencję zarówno dla pojedynczego klienta, jak i seryjną do kontrahentów. Mamy możliwość prowadzenia kilku rejestrów VAT, rejestru sprzedaży i zakupów. Ponadto wydrukujemy faktury, dokonamy przelewów oraz rozliczymy koszty z podziałem na budynki.
- **Kasa:** Program Kasa daje możliwości prowadzenia kasy w biurze. Ułatwia rejestrację operacji kasowych dzięki wypełnianiu odpowiednich dokumentów KW, KP i innych. Umożliwi wykonanie raportów kasowych i bankowych z opcją wydruku. Programy FK oraz Czynsze współpracują z programem KASA, dzięki czemu kasjerka widzi aktualne stany konta wpłacającego. Możliwości obsługi wielu kas daje możliwość utworzenia nieograniczonej liczby stanowisk.
- **ADA WEB:** Program przeznaczony dla lokatorów. Pozwala na przeglądanie stanu kont przez przeglądarkę internetową. Dostęp do konta jest chroniony hasłem nadanym przez administratora.
- **Zgłoszenia:** Możliwość zgłaszania opinii i awarii do działu administracyjno-technicznego. Przechowuje listę zgłoszeń, umożliwia kontrolę terminu i sposobu realizacji.

3.2. Wirtualne Osiedle

Kolejnym systemem, który jest dostępny i oferuje podobny zakres usług, jest http://wirtualneosiedle.pl/. System ten został podzielony na trzy moduły:

3.2.1 Księgowość wspólnot

Księgowość (Rysunek 4)– moduł stworzony dla administratorów i księgowych.

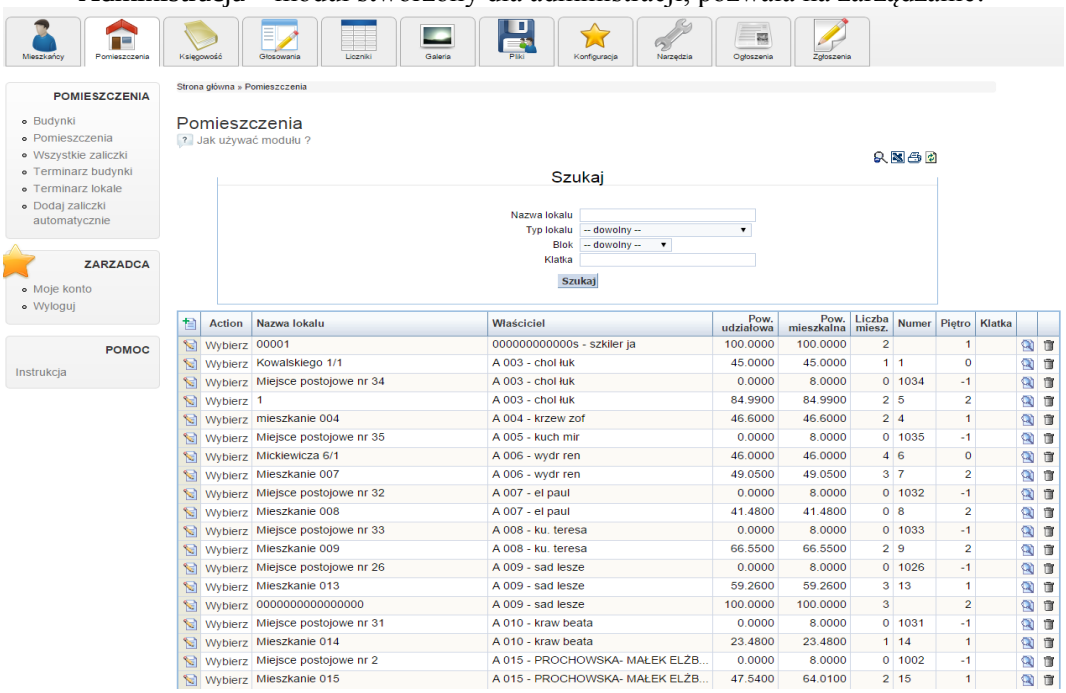


Rysunek 4. Moduł księgowość. ***Źródło: Opracowanie własne.***

- **Integracja z bankiem:** Program posiada opcję integracji z każdym dostępnym na rynku systemem bankowym. Polega to na łączeniu się z bankiem i automatycznym księgowaniu. Dzięki takiemu rozwiązaniu dane są samoczynnie sczytywane z wyciągów cyfrowych. Możliwa jest także współpraca z subkontami oraz księgowanie po tytule płatności, co znacząco zmniejsza koszty utrzymania. System ma wbudowane zabezpieczenia, które uniemożliwiają przypadkowe wprowadzenie identycznych danych.
- **Automatyzacja pracy:** System automatycznie powiadamia mieszkańców o zmianie stanu konta oraz o zaległościach. Takie rozwiązanie jest korzystne dla mieszkańców, a jednocześnie zmniejsza problemy administratora wynikające z nieterminowych płatności, których powodem często jest zapominalstwo. Dostęp on-line pozwala właścicielowi na przeanalizowanie zawsze aktualnych naliczeń i wpłat,. Znacząco ogranicza to liczbę zapytań i nieporozumień. Wyciągi możemy zapisywać w pliku PDF, który jest generowany na żądanie. Udostępniono także funkcję masowego przesyłania e-maili do mieszkańców z informacją o saldzie konta.
- **Dostęp 24-godziny 7 dni w tygodniu:** Dostęp do systemu dzięki zastosowaniu podejścia klient – serwer jest niezwykle prosty. Jedynym wymaganiem jest komputer z dowolną przeglądarką, aby móc zalogować się do Wirtualnego Osiedla. Dane są zapisywane na serwerze, zatem w razie awarii wystarczy zalogować się na innym komputerze, dzięki czemu unika się przerwy w pracy bądź, co gorsza, utraty danych. Dodatkowym atutem jest możliwość pracy w dowolnym miejscu, w dowolnym czasie.
- **Bezpieczeństwo:** Korzystając z Wirtualnego Osiedla, nie musimy się obawiać utraty danych spowodowanej awarią dysków twardych, które, jak wiadomo, posiadają ograniczoną wytrzymałość. Automatyczne kopie zapasowe oraz szyfrowane połączenie w stu procentach zapewniają trwałość oraz poufność danych.

3.2.2 Administracja

Administracja – moduł stworzony dla administracji, pozwala na zarządzanie:



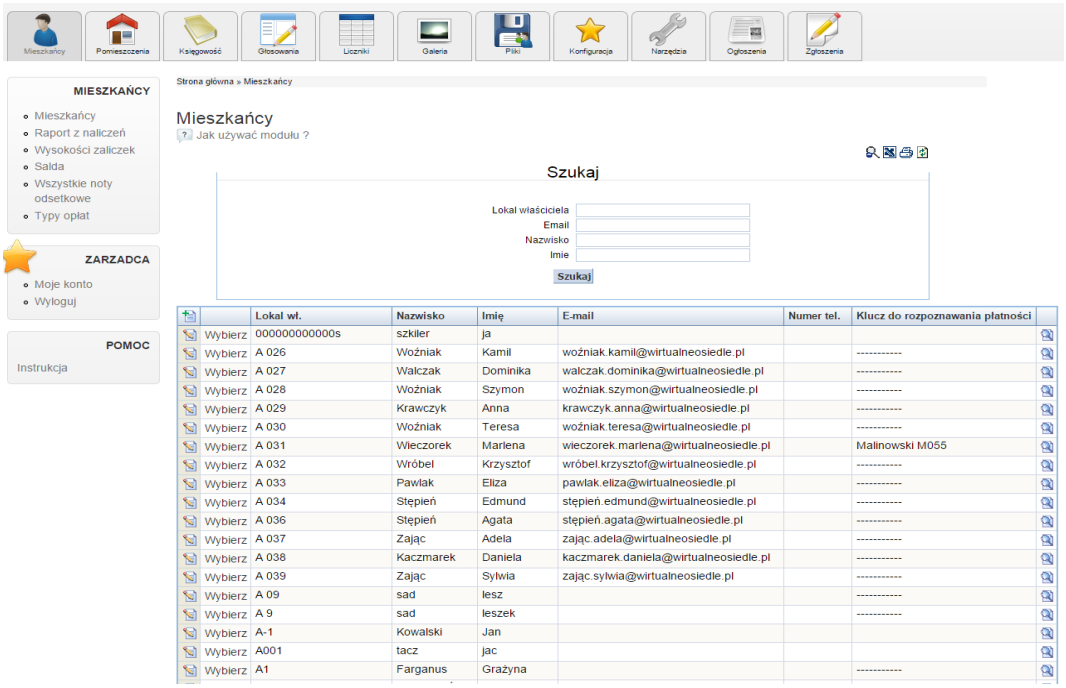
Action	Nazwa lokalu	Właściciel	Pow. udziałowa	Pow. mieszkalna	Liczba miesz.	Numer	Piętro	Klatka
Wybierz	00001	000000000000s - szkiler ja	100.0000	100.0000	2		1	
Wybierz	Kowalskiego 1/1	A 003 - chol łuk	45.0000	45.0000	1	1	0	
Wybierz	Miejsce postojowe nr 34	A 003 - chol łuk	0.0000	8.0000	0	1034	-1	
Wybierz	1	A 003 - chol łuk	84.9900	84.9900	2	5	2	
Wybierz	mieszkanie 004	A 004 - krzew zof	46.6000	46.6000	2	4	1	
Wybierz	Miejsce postojowe nr 35	A 005 - kuch mir	0.0000	8.0000	0	1035	-1	
Wybierz	Mickiewicza 6/1	A 006 - wydr ren	46.0000	46.0000	4	6	0	
Wybierz	Mieszkanie 007	A 006 - wydr ren	49.0500	49.0500	3	7	2	
Wybierz	Miejsce postojowe nr 32	A 007 - el paul	0.0000	8.0000	0	1032	-1	
Wybierz	Mieszkanie 008	A 007 - el paul	41.4800	41.4800	0	8	2	
Wybierz	Miejsce postojowe nr 33	A 008 - ku. teresa	0.0000	8.0000	0	1033	-1	
Wybierz	Mieszkanie 009	A 008 - ku. teresa	66.5500	66.5500	2	9	2	
Wybierz	Miejsce postojowe nr 26	A 009 - sad lesze	0.0000	8.0000	0	1026	-1	
Wybierz	Mieszkanie 013	A 009 - sad lesze	59.2600	59.2600	3	13	1	
Wybierz	0000000000000000	A 009 - sad lesze	100.0000	100.0000	3		2	
Wybierz	Miejsce postojowe nr 31	A 010 - kraw beata	0.0000	8.0000	0	1031	-1	
Wybierz	Mieszkanie 014	A 010 - kraw beata	23.4800	23.4800	1	14	1	
Wybierz	Miejsce postojowe nr 2	A 015 - PROCHOWSKA- MAŁEK ELŻB...	0.0000	8.0000	0	1002	-1	
Wybierz	Mieszkanie 015	A 015 - PROCHOWSKA- MAŁEK ELŻB...	47.5400	64.0100	2	15	1	

Rysunek 5. Zakładka pomieszczenia. ***Źródło: Opracowanie własne.***

- **Ewidencja mieszkańców:** Mieszkańcy są przedstawieni za pomocą listy w sposób bardzo jasny i czytelny. Bezpośredni dostęp z listy do karty mieszkańca pozwala w każdej chwili na wgląd do podstawowych danych. W karcie mieszkańca znajdziemy dane kontaktowe, liczniki, głosowania, saldo, emaile, sms-y. Specjalna strefa mieszkańca pozwala na wspólne rozwiązywanie problemów bez konieczności spotykania się. Administrator i mieszkaniec podczas rozmowy w jednym czasie mają dostęp do identycznych danych.
- **Komunikacja z mieszkańcem:** Dobra komunikacja jest podstawą sprawnie działającego biznesu. Administrator za pomocą grupowych e-maili i sms-ów na bieżąco informuje o terminie płatności czy remoncie, który utrudni funkcjonowanie mieszkańców.
- **Głosowania:** Zastosowanie elektronicznej karty do głosowania znacząco zwiększa frekwencję. Wystarczy odebrać e-mail z kartą i kliknąć wybraną opcję. Oszczędza to czas mieszkańca oraz umożliwia uczestnictwo w głosowaniu na poziomie 95%.
- **Liczniki:** Dzięki ewidencji liczników każdy mieszkaniec ma dostęp do swoich pomiarów, także księgowi mają możliwość dokonania rozliczeń w prosty sposób.
- **Integralność danych:** Wszyscy użytkownicy systemu pracują na jednej bazie, dzięki czemu nie ma problemu rozbieżności danych. Bezpieczeństwo danych zapewnia różny poziom dostępu.

3.2.3 Mieszkańcy

Mieszkańcy (Rysunek 6)– moduł stworzony dla mieszkańców, pozwala na zarządzanie kontem:

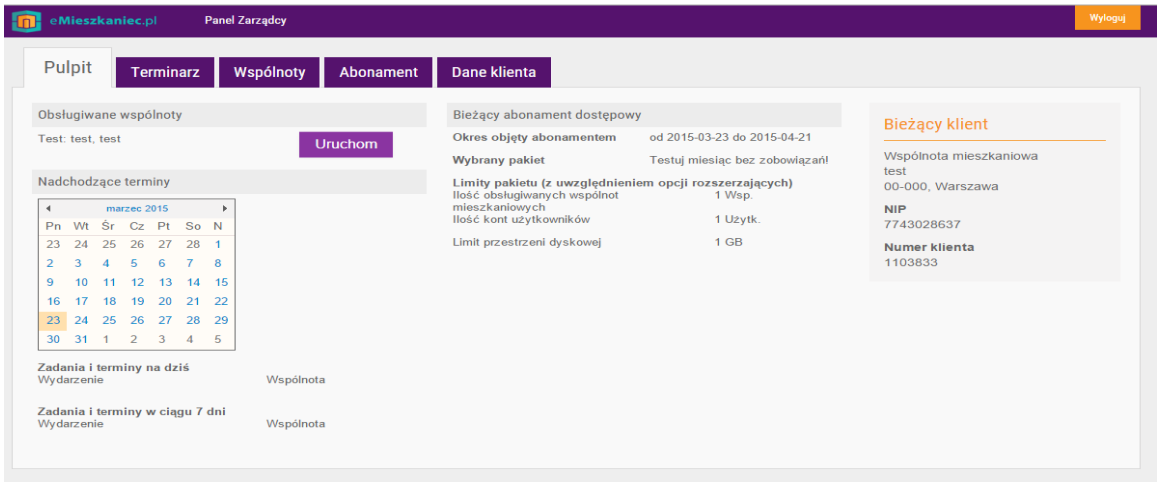


Rysunek 6. Moduł mieszkańcy. *Źródło: Opracowanie własne.*

Internetowe konto. Aktualne saldo, wpłaty i zobowiązanie – wszystko to może sprawdzić i przeanalizować właściciel mieszkania. Udostępniony jest szczegółowy wykaz wszystkich sald, naliczeń i wpłat oraz historia odczytów liczników, głosowań, e-maili i sms-ów. Zarządca korzystając z serwisu może udostępniać pliki lub strony internetowe z informacjami do zapoznania się mieszkańcom. Każdy użytkownik posiada indywidualny login i hasło, które zabezpieczają przed dostępem osób niepowołanych. Po zalogowaniu uzyskuje się dostęp wyłącznie do swoich zasobów.

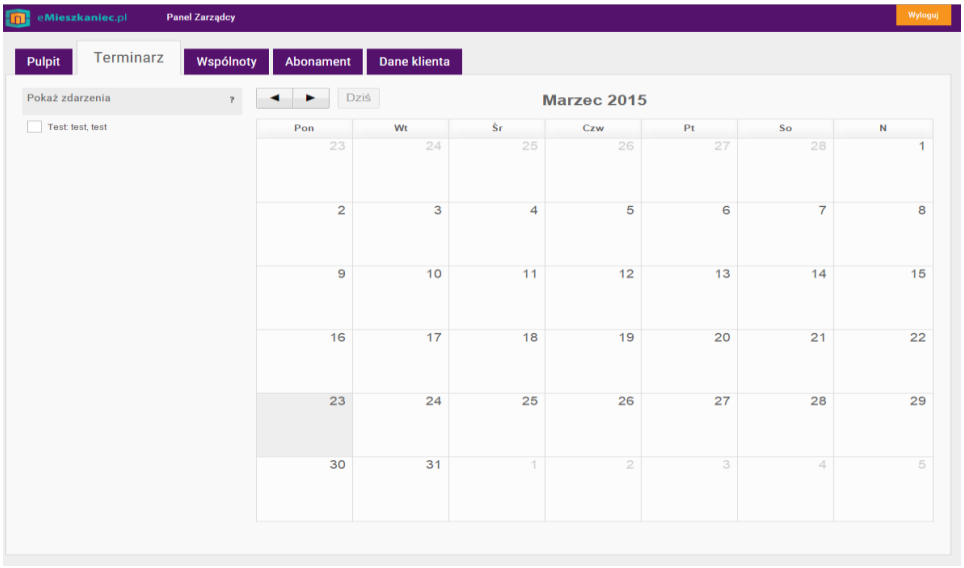
3.3. E-mieszkaniec.pl

Kolejnym rozwiązaniem dostępnym na rynku jest E-mieszkaniec. System jest dostępny z poziomu przeglądarki, co znacząco ułatwia pracę. Po zalogowaniu się do systemu trafiamy na pulpit panelu zarządcy – Rysunek 7.



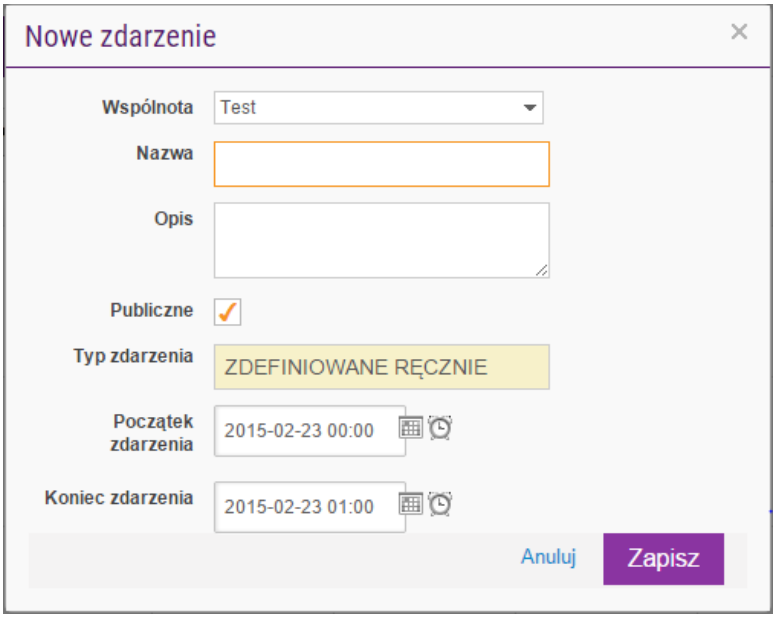
Rysunek 7. Panel Zarządcy E-mieszkaniec. *Źródło: Opracowanie własne.*

Znajdziemy tu mały kalendarz, informacje o abonamencie, obsługiwane wspólnoty oraz informacje o bieżącym kliencie. Kolejna zakładka to terminarz – Rysunek 8.



Rysunek 8. Terminarz panelu zarządcy. *Źródło: Opracowanie własne.*

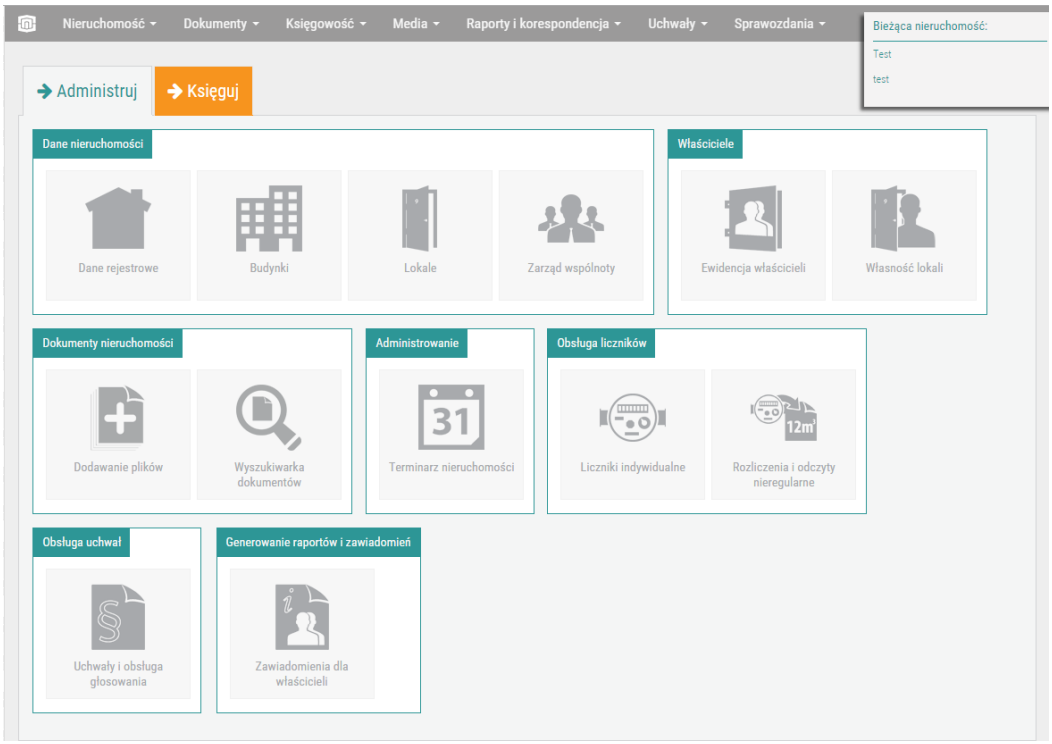
Widnieje tu prosty kalendarz, po kliknięciu w datę pojawia się okno, gdzie możemy zapisać dane wydarzenie – Rysunek 9.



Rysunek 9. Dodawanie nowego zdarzenia. *Źródło: Opracowanie własne.*

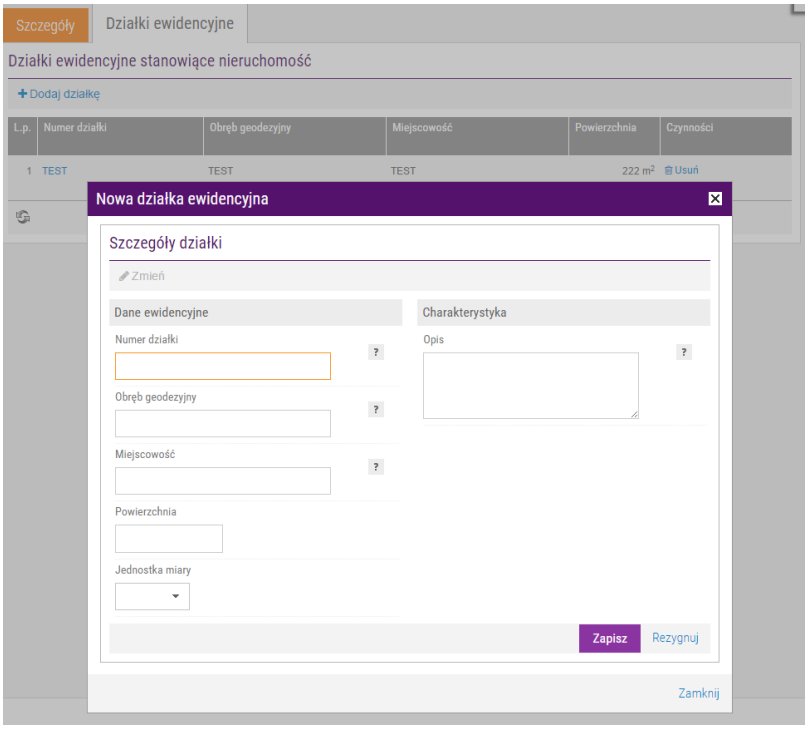
3.3.1 Panel Administratora

Po przejściu do panelu administratora wspólnoty przechodzimy na pulpit główny wspólnoty – Rysunek 10.



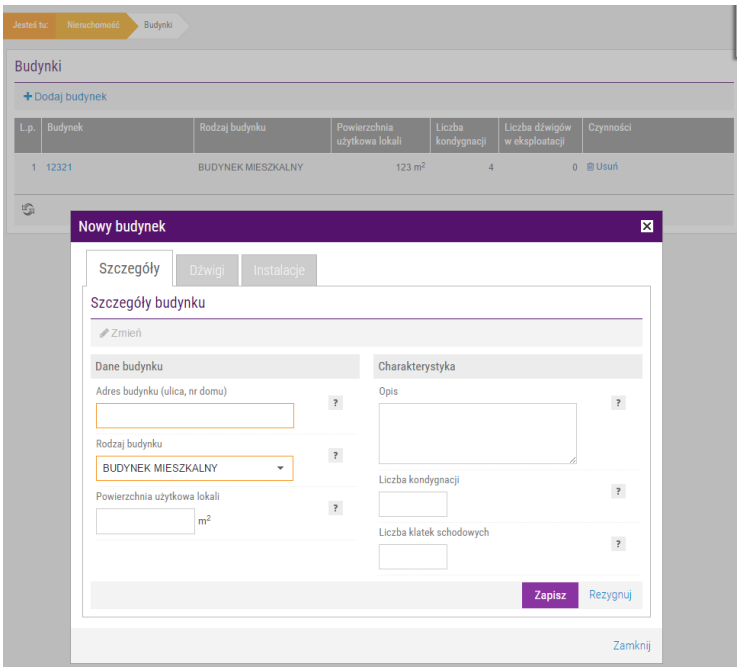
Rysunek 10. Pulpit główny w panelu administratora. ***Źródło: Opracowanie własne.***

W panelu administratora mamy zakładkę dane rejestrowe (Rysunek 11), zawierającej takie informacje, jak powierzchnia działki, ilość budynków i mieszkań w budynkach.



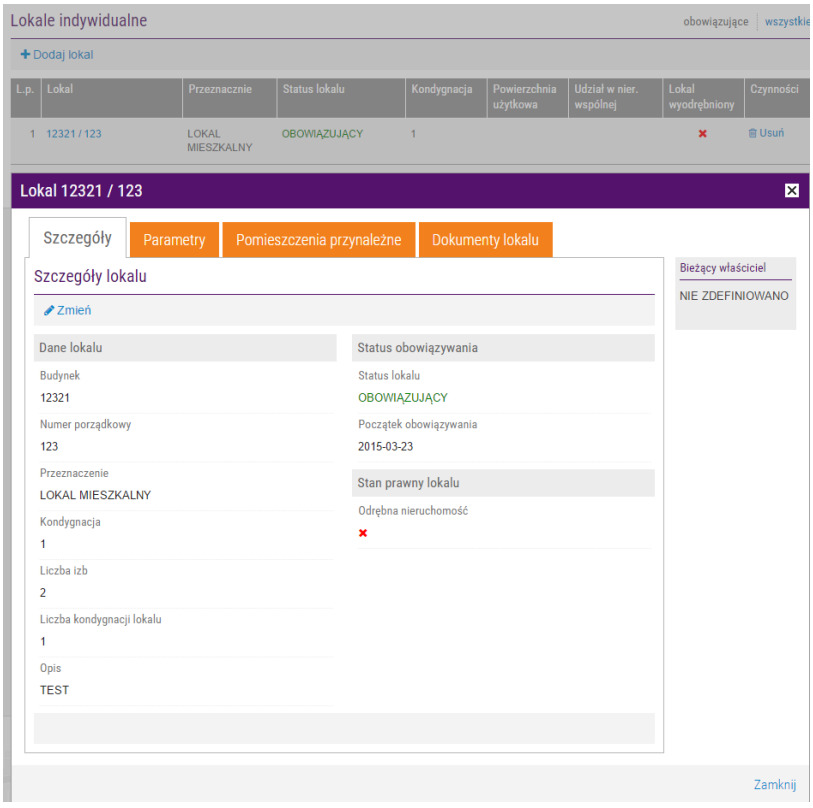
Rysunek 11. Dane Rejestrowe. ***Źródło: Opracowanie własne.***

Kolejna zakładka to Budynki. Możemy dzięki niej dodać, edytować i wyświetlić budynki – Rysunek 12.

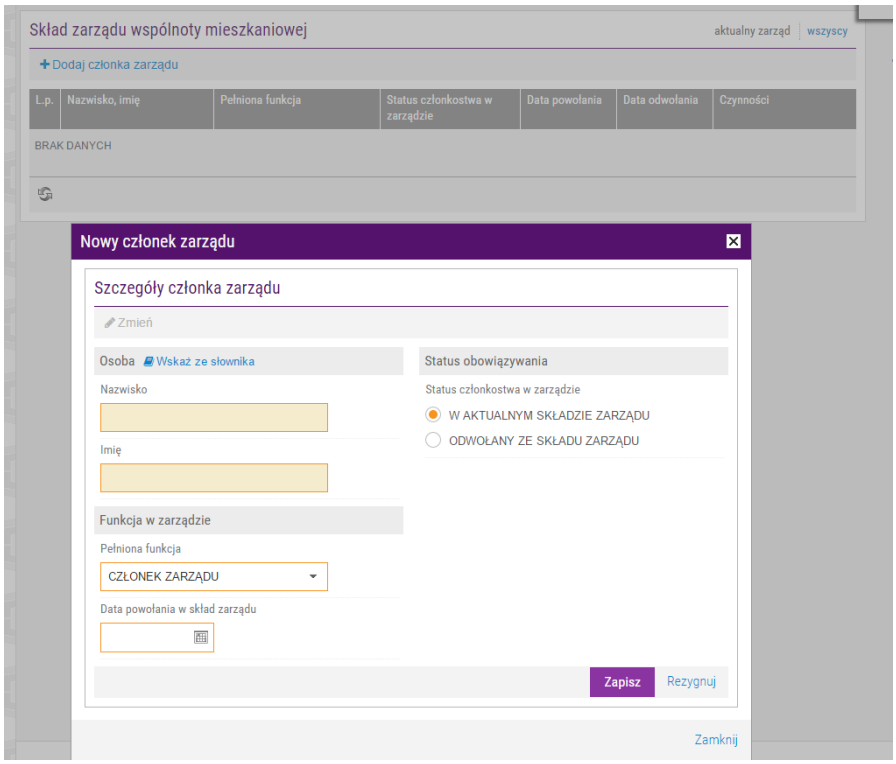


Rysunek 12. Zakładka Budynki. ***Źródło: Opracowanie własne.***

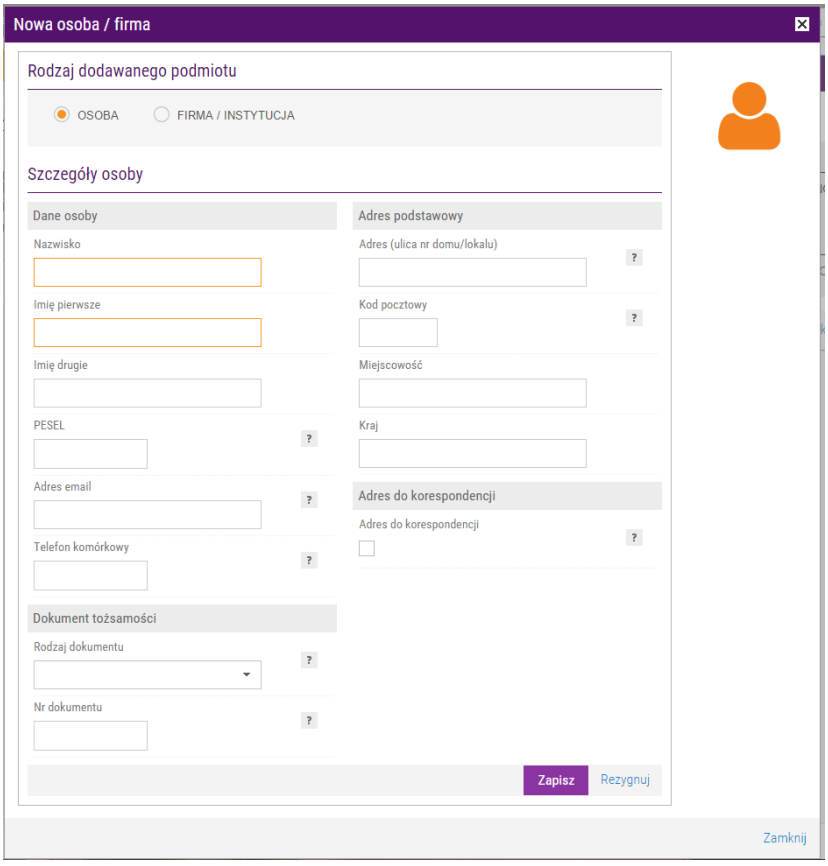
Ostatnie zakładki to lokale (Rysunek 13), właściciele (Rysunek 15) oraz skład zarządu spółdzielni (Rysunek 14). Pierwsza z nich zawiera podstawowe informacje o lokalach i należących do nich pomieszczeniach oraz dokumenty i parametry. W zakładce składu zarządu spółdzielni możemy zarówno dodać, jak i odwołać członka zarządu.



Rysunek 13. Zakładka Lokale. ***Źródło: Opracowanie własne.***



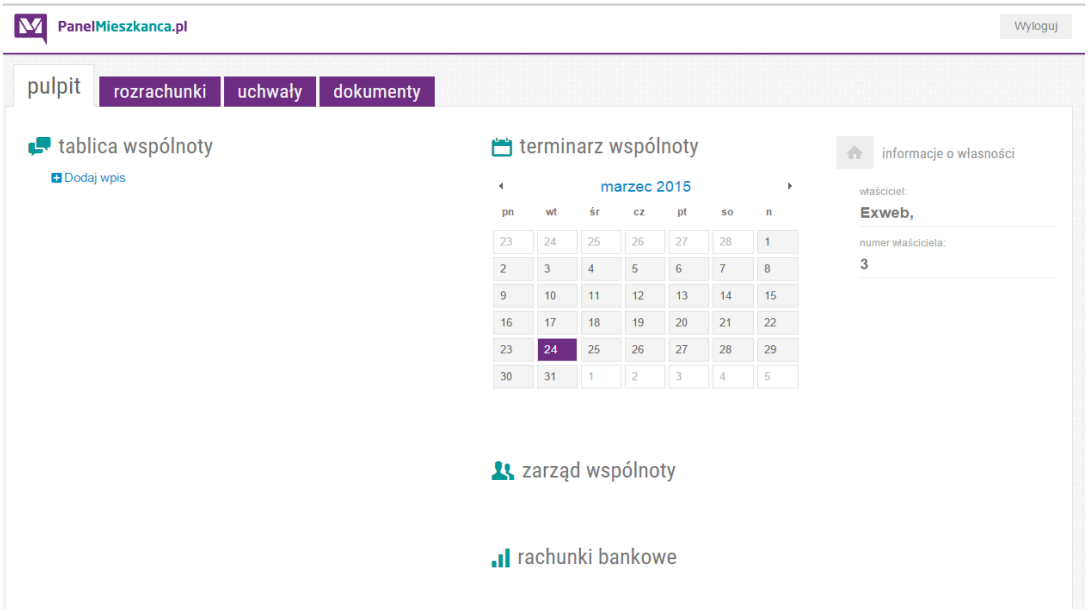
Rysunek 14. Zakładka skład zarządu. ***Źródło: Opracowanie własne.***



Rysunek 15. Lista osób/firm, które są właścicielami lokali. ***Źródło: Opracowanie własne.***

3.3.2 Panel Mieszkańca

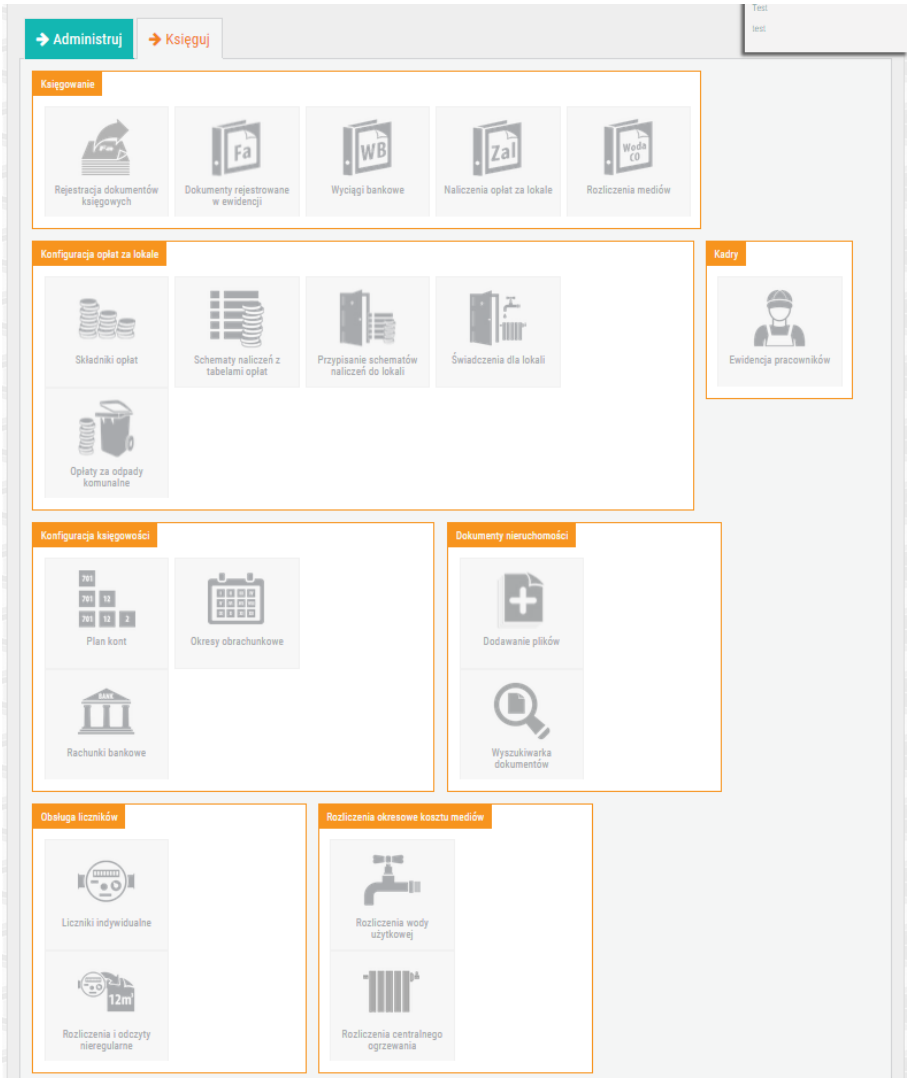
Panel mieszkańca (Rysunek 16) – właściciela lokalu. Znajdziemy tam terminarz wspólnoty, tablicę ogłoszeń, informacje o zarządzie i rachunku bankowym.



Rysunek 16. Panel Mieszkańca. ***Źródło: Opracowanie własne.***

3.3.3 Panel Księgowy

Panel księgowy (Rysunek 17) – dzięki niemu możemy zarządzać księgowaniem wpłat, rozliczać opłaty za lokale, skonfigurować księgowość i przetrzymywać dokumenty nieruchomości, a także zarządzać kadrą.



Rysunek 17. Panel Księgowy. ***Źródło: Opracowanie własne.***

System jest intuicyjny, łatwy w obsłudze, aplikacje są dostępne przez przeglądarkę bez konieczności instalacji. Podobnych systemów jest na naszym rynku wiele, np. Weles3, axxerion i wiele innych, które posiadają podobne funkcjonalności.

3.4. Podsumowanie

Wszystkie wyżej wymienione aplikacje posiadają jedną wspólną wadę. Są systemami, które ograniczają się tylko do zarządzania pojedynczymi wspólnotami w zakresie administracyjno-finansowym. Próżno w nich szukać takich funkcjonalności, jak raporty administratora czy zgłaszanie pomysłów inwestycji, a jak wiadomo, każdy chce wiedzieć, co się dzieje w jego miejscu zamieszkania, i móc współuczestniczyć w podejmowaniu decyzji, które tego miejsca dotyczą. Brakuje także wyodrębnienia lokatora i właściciela mieszkania, co naszym zdaniem jest w dzisiejszych czasach niezbędne, ponieważ jak podają źródła [2] około 4,2% Polaków decyduje się na wynajem. Tworzenie jednego konta dla jednego mieszkania pozbawia osoby wynajmujące dostępu do podstawowych informacji, takich jak ogłoszenia czy informacje o awariach. Właściciel nie zawsze bowiem chce przekazać swoje konto najemcy. Często są tam udostępniane informacje, których nie chce ujawniać, np. stan konta mieszkania, wykaz płatności czy też posiadanie większej ilości obiektów.

4. Propozycja systemu zarządzania wspólnotami mieszkaniowymi

W niniejszym rozdziale zostaną opisane i przedstawione wszystkie funkcjonalności systemu zarządzania wspólnotami.

4.1. Nawigacja

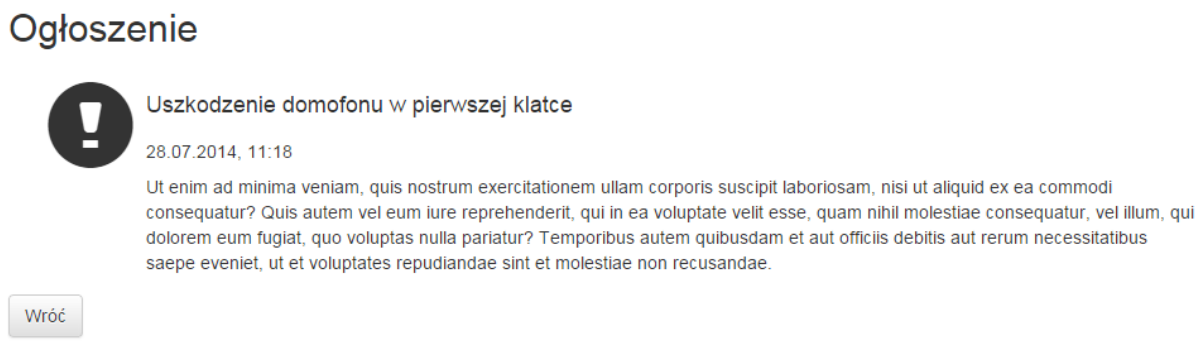
Wygląd aplikacji jest jedną z najważniejszych cech wytwarzanego oprogramowania. W dobie, gdzie podobnych rozwiązań jest dużo, użytkownik wybierze tą, która mu najbardziej odpowiada pod względem łatwego użytkowania. Starano się zadbać o to, aby cała aplikacja wyglądała i była obsługiwana w ten sam sposób. Zadbano także o responsywność. Ważnym elementem jest fakt, iż grupa docelowa użytkowników jest zróżnicowana wiekowo, a co się z tym wiąże, posiada rożny stopień umiejętności obsługi komputera. Przejrzystość i intuicyjność starano się osiągnąć poprzez nawiązanie do istniejących zwyczajów dotyczących GUI, które przedstawimy i opiszemy poniżej.

Aby móc w pełni korzystać z funkcjonalności, które udostępnia SZW[1], konieczne jest logowanie. Rysunek 18 przedstawia panel logowania oraz ogłoszenia, które są ogólnodostępne.

Rysunek 18. Strona logowania. ***Źródło: Opracowanie własne.***

Treść ogłoszeń została ograniczona, aby poprawić czytelność strony. Należy kliknąć w temat ogłoszenia, aby przejść do pełnego opisu (Rysunek 19).

[1] Skrót pierwszych liter od **S**ystemu **Z**arządzania **W**spólnotą



Rysunek 19. Widok ogłoszeń. ***Źródło: Opracowanie własne.***

Konto jest zakładane przez administratora systemu . Wyróżniono kilka rodzajów kont:

- lokator,
- właściciel,
- zarządca,
- członek zarządu,
- księgowość,
- administrator systemu.

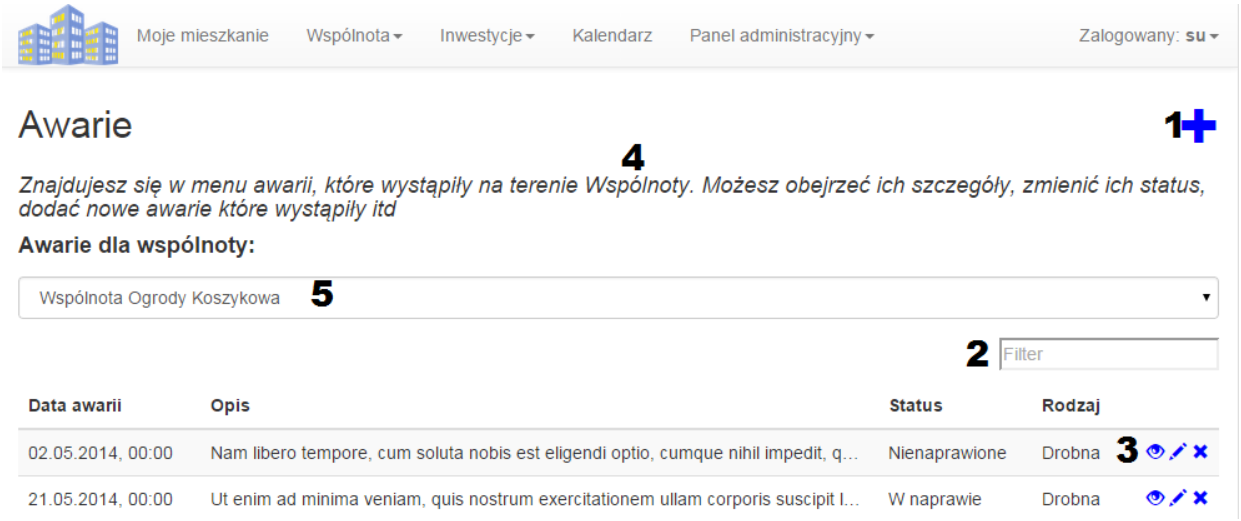
Każde z nich ma przypisane uprawnienia, wynikające z diagramu przypadków użycia (77). Aby zaprezentować wygląd, rozwiązania i funkcjonalność całego systemu, będziemy korzystać z konta super usera.

Odpowiednia nawigacja to fundament każdej dobrze prosperującej strony. Jej zadaniem jest prowadzenie odwiedzających tak, aby nie potrzebowali dodatkowej pomocy. Ma to duży wpływ na szansę odnalezienia potrzebnych informacji, czas ich wyszukiwania oraz, co najważniejsze, zadowolenie użytkownika. Z doświadczenia wiemy, iż w większości przypadków najlepiej sprawdzają się proste, intuicyjne i konwencjonalne rozwiązania.

Nawigacja została podzielona na 5 głównych bloków tematycznych: Moje mieszkanie, Wspólnota, Inwestycje, Kalendarz, Panel administracyjny. Zastosowano tu jedną z najpopularniejszych form nawigacji – „Drop-Down". Zasada funkcjonowania jest adekwatna do logo Windows w systemie operacyjnym. Należy kliknąć myszką w blok tematyczny, aby zobaczyć, co znajduje się w podkategoriach. Takie podejście zaowocowało posegregowanymi i uporządkowanymi tematycznie danymi. Rysunek 20 przedstawia menu drop-down zaprojektowane na potrzeby projektu.

Rysunek 20. Nawigacja SZW. ***Źródło: Opracowanie własne.***

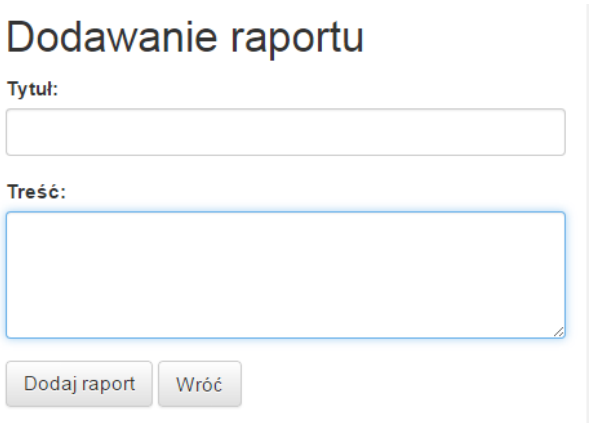
We wszystkich przypadkach w których zachodziła taka konieczność zaimplementowane zostało filtrowanie (pkt. 2 - Rysunek 21) co niewątpliwie ułatwiło przeszukiwanie tabeli według kryteriów zdefiniowanych przez użytkownika. Dodawanie obiektów zostało we wszystkich miejscach oznaczone dużym niebieskim znakiem „+” (pkt. 1 - Rysunek 21), a wszystkie ikony symbolizujące akcje zostały dodane do każdego wiersza z wyrównaniem do prawej strony. Rysunek 21 przedstawia wygląd ikon z ich kolejnością (pkt. 3), i tak od lewej strony mamy ikonę „Szczegóły”, „Edycja” i „Usuwanie”. Dodatkowo po najechaniu na każdą z nich wyświetla się chmurka z opisem.



Rysunek 21. Elementy nawigacji w projekcie. ***Źródło: Opracowanie własne.***

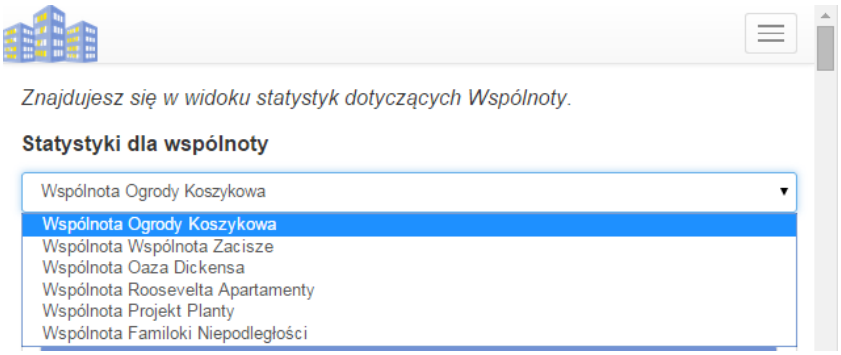
Elementem wprowadzenia do każdej funkcjonalności jest krótki opis znajdujący się zawsze pod tytułowym nagłówkiem (pkt. 4 - Rysunek 21). We wszystkich miejscach, gdzie było konieczne filtrowanie po wspólnotach zastosowano rozwijaną listę

Wszystkie aktywne pola formularzy posiadają niebieską ramkę, dlatego nawet najmniej zaawansowani użytkownicy nie powinni mieć problemu z dodawaniem nowych treści. Rozwiązanie przedstawiono za pomocą formularza dodawania nowego raportu (Rysunek 22).



Rysunek 22. Aktywny element formularza. ***Źródło: Opracowanie własne.***

Rozwiązaniem ułatwiającym sprawne przeglądanie dużej ilości danych jest wyświetlanie po wspólnotach dodanych do systemu we wszystkich elementach, gdzie zachodziła taka konieczność. Rysunek 21 pkt. 5 oraz Rysunek 23 przedstawiają wygląd rozwiązania.



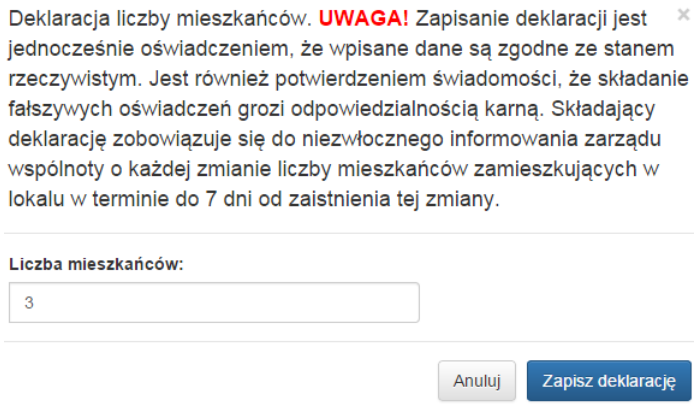
Rysunek 23. Filtrowanie wspólnot. *Źródło: Opracowanie własne.*

4.2. Moje mieszkanie

Jednym z najważniejszych elementów projektowanego systemu jest widok „Moje mieszkanie" (Rysunek 24). Znajdziemy w nim podstawowe dane naszego lokalu, takie jak adres, metraż, piętro, liczbę mieszkańców, wysokość czynszu. Dane te zostały zaprezentowane za pomocą kafli, które są według nas czytelne, nowoczesne i funkcjonalne.

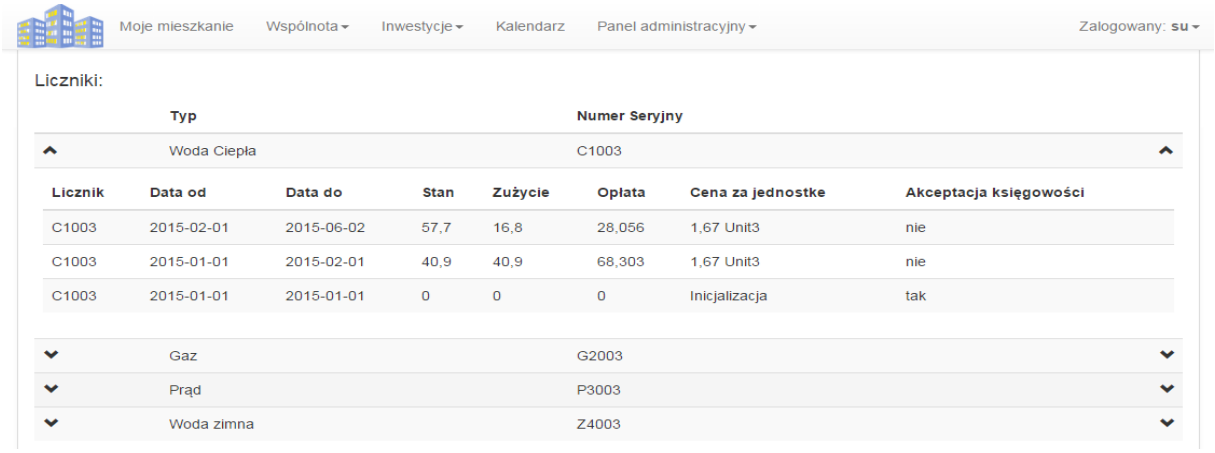
Rysunek 24. Moje mieszkanie cz.1 - Informacje. *Źródło: Opracowanie własne.*

Z poziomu mieszkania dostępna jest także deklaracja liczby mieszkańców. Aby jej dokonać, wystarczy kliknąć w „Zadeklaruj liczbę mieszkańców", po czym pojawi się okno przedstawione poniżej (Rysunek 25).



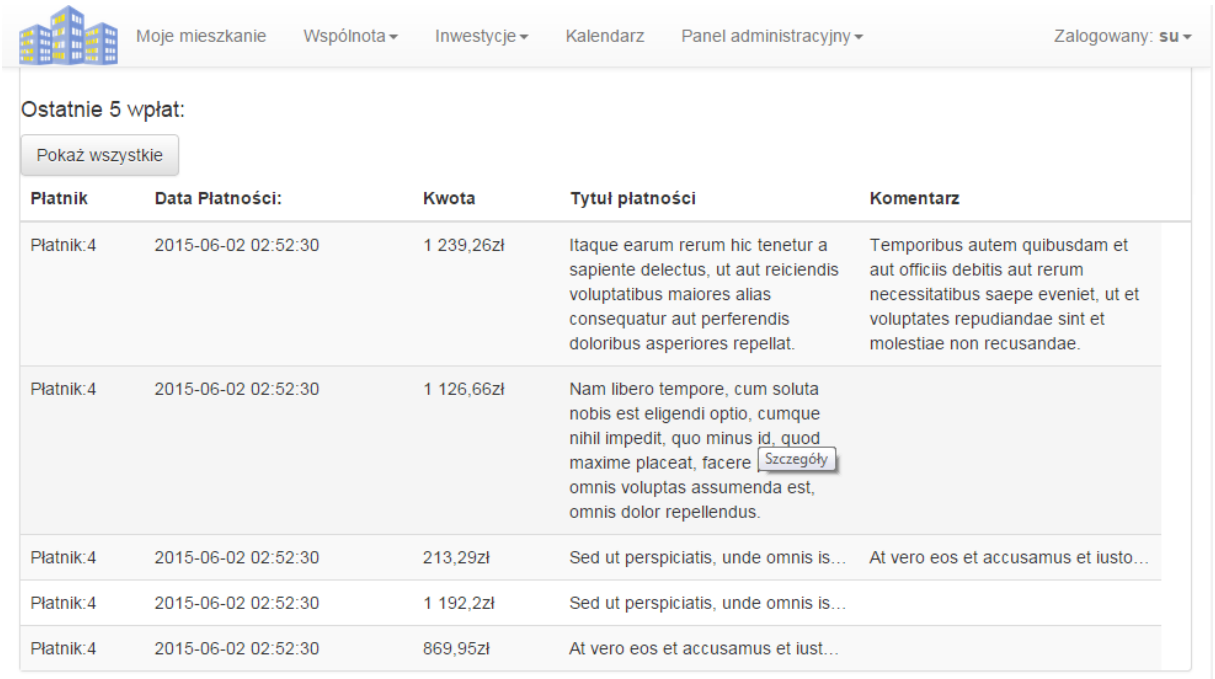
Rysunek 25. Deklaracja liczby mieszkańców. *Źródło: Opracowanie własne.*

Dzięki dostępnemu modułowi „Liczniki” użytkownik ma podgląd na zegary zużycia mediów w swoim mieszkaniu. Może zobaczyć ich typ i numer oraz szczegóły, które zawierają podstawowe informacje, takie jak okres, zużycie, cena.



Rysunek 26. Moje mieszkanie cz.2 - Liczniki. ***Źródło: Opracowanie własne.***

Wyświetlane są także wpłaty – domyślnie 5 ostatnich. Za pomocą przycisku „Pokaż wszystkie” wyświetlona zostaje pełna lista płatności. Dla czytelności reprezentowanych danych postanowiono ograniczyć długość tytułu płatności oraz komentarz. Aby mieć podgląd na całość wystarczy kliknąć w interesującą nas płatność. Rysunek 27 przedstawia zastosowane rozwiązanie.



Rysunek 27. Moje mieszkanie cz.3 - Wpłaty. ***Źródło: Opracowanie własne.***

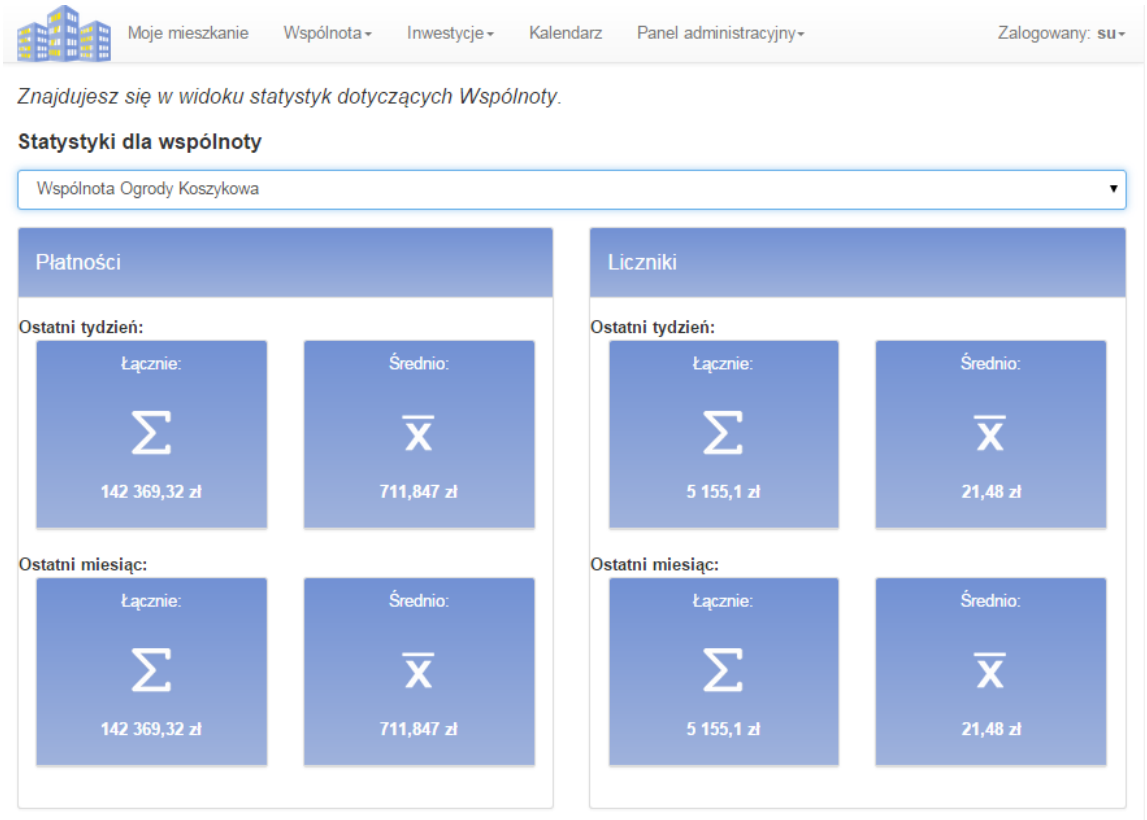
Podsumowując „Moje mieszkanie” zawiera wszystkie dane dotyczące mieszkania w jednym miejscu. Informacje te są ograniczone (ilościowo) oraz pogrupowane. Jest to optymalne rozwiązanie które z pewnością nie sprawi problemów użytkownikowi, a jednocześnie zapewni przejrzystość i łatwość w użytkowaniu.

4.3. Wspólnota

Wszystkie informacje dotyczące wspólnoty mieszkaniowej znajdują się w zakładce „Wspólnota”. Funkcjonalność, wygląd i przeznaczenie każdej z nich zostanie opisane w poniższych punktach.

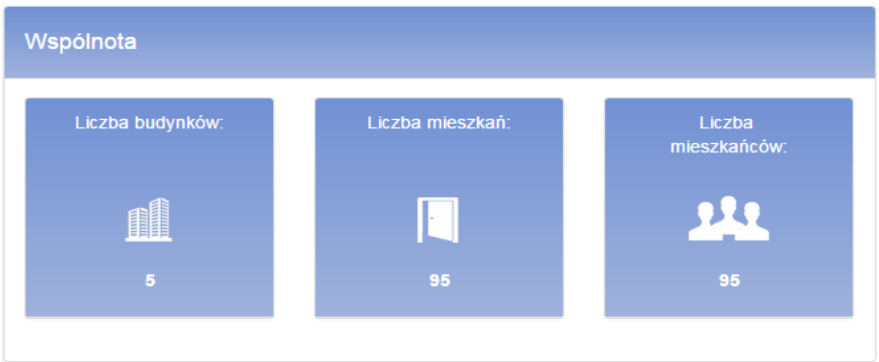
4.3.1 Statystyki

W statystykach (Rysunek 28) znajdują się informacje dotyczące wyników finansowych wspólnoty. Zostały one podzielone na tygodniowy oraz miesięczny okres rozliczeniowy, z podziałem na łączny i średni koszt podany w złotówkach. Elementy, których statystki dotyczą, to płatności oraz liczniki.



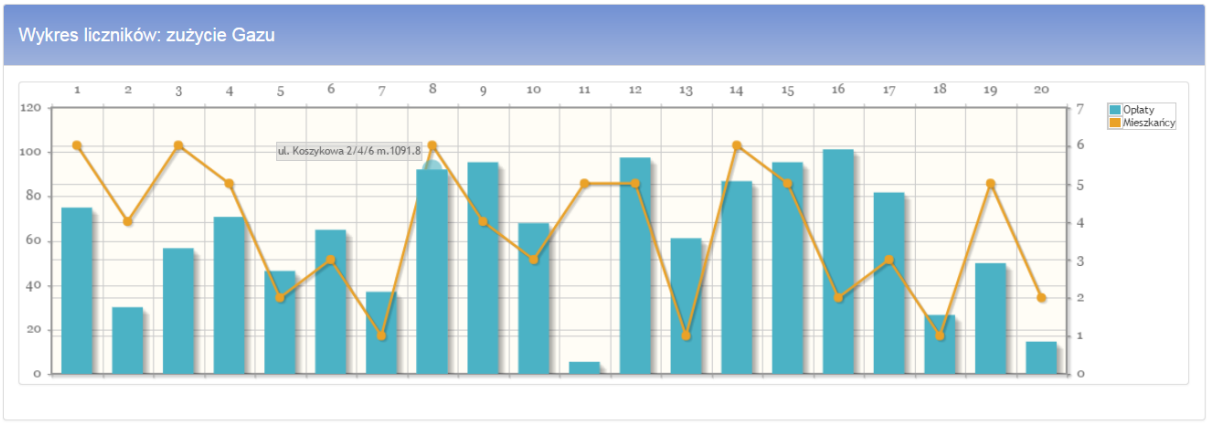
Rysunek 28. Statystyki cz.1 – Płatności i Liczniki. *Źródło: Opracowanie własne.*

Zaimplementowane zostały także statystyki dla wspólnoty, których zawartość jest przedstawiona na Rysunek 29**.**

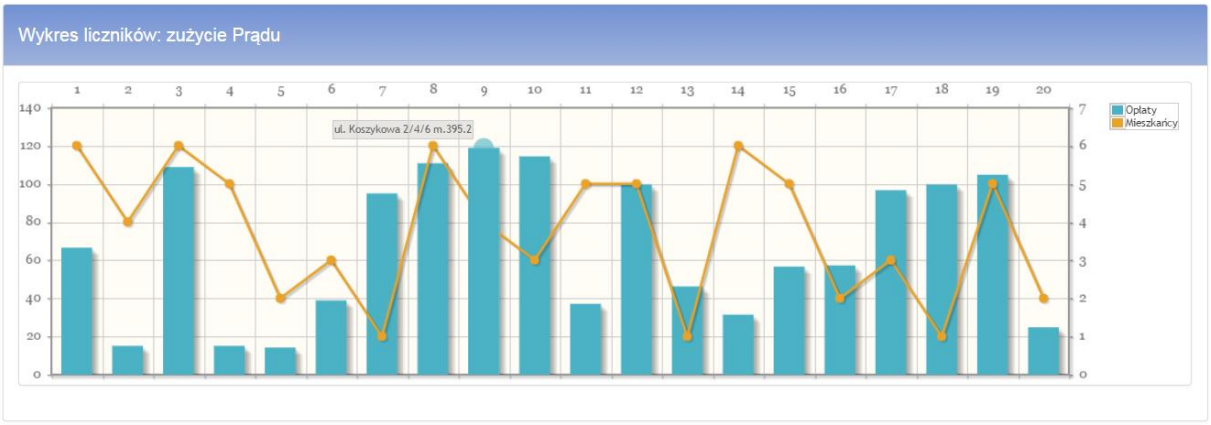


Rysunek 29. Statystyki wspólnoty. *Źródło: Opracowanie własne.*

Dodatkowym atutem są wykresy, który wizualnie przedstawiają zużycie takich mediów jak gaz, prąd, woda zimna i ciepła (Rysunek 30, Rysunek 31). Wykres przedstawia wysokość opłat oraz liczbę mieszkanców danej ulicy. Adres jest widoczny po najechaniu kusorem na słupek wykresu.



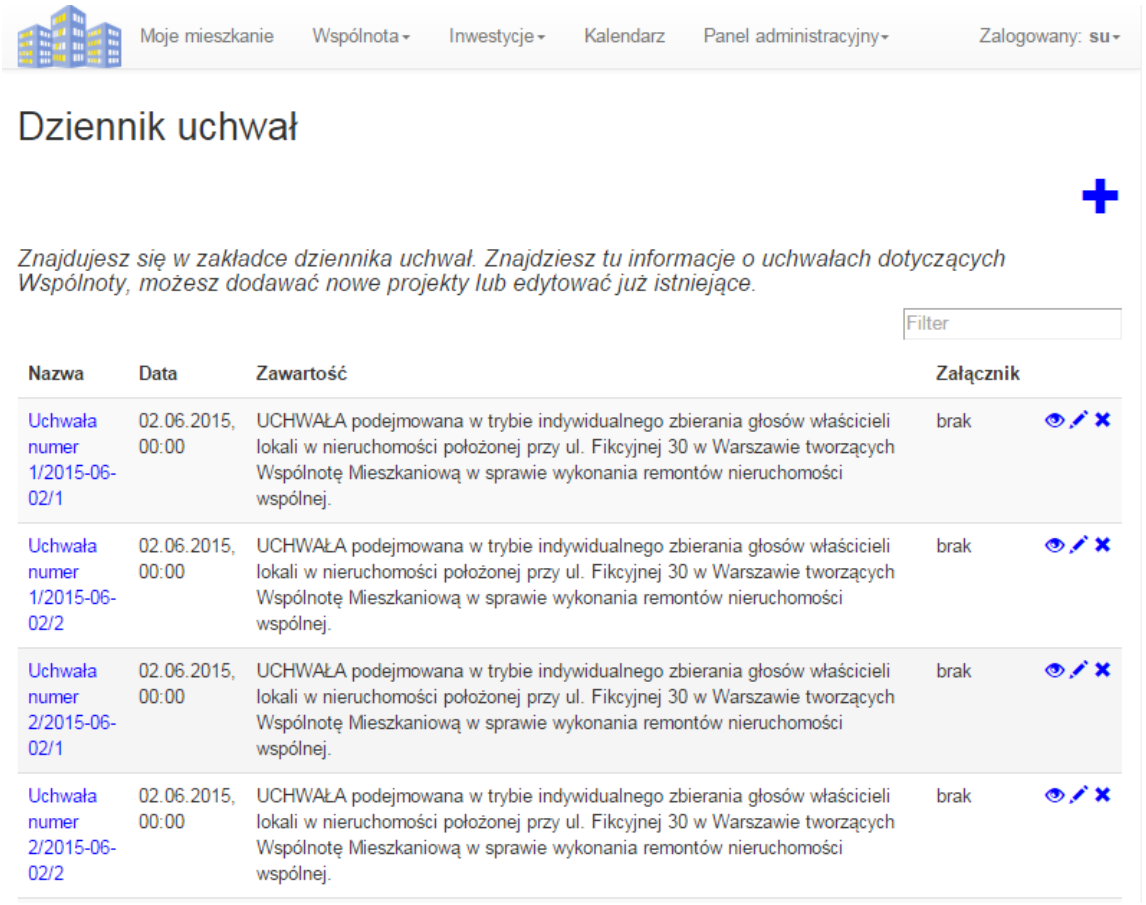
Rysunek 30. Wykres płatności - gaz. ***Źródło: Opracowanie własne.***



Rysunek 31. Wykres płatności - prąd. Źródło: Opracowanie własne.

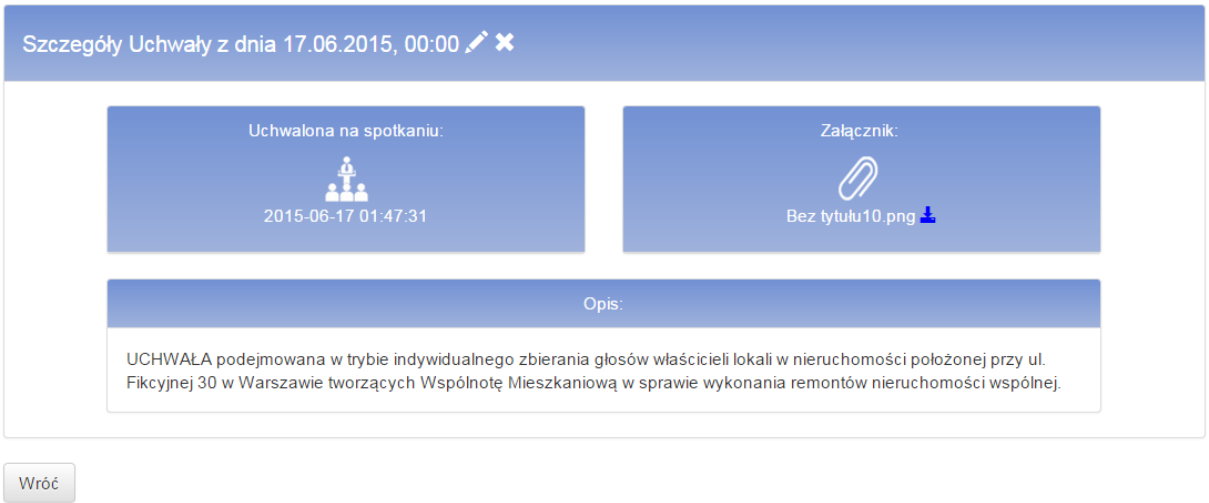
4.3.2 Uchwały

Kolejnym elementem wspólnoty są uchwały. Aby poprawić czytelność i łatwość użytkowania, dziennik został przedstawiony za pomocą tabeli z odpowiednimi opisami kolumn (Rysunek 32). Zastosowano także wspomniane wyżej filtrowanie oraz ikony akcji.



Rysunek 32. Dziennik Uchwał. ***Źródło: Opracowanie własne.***

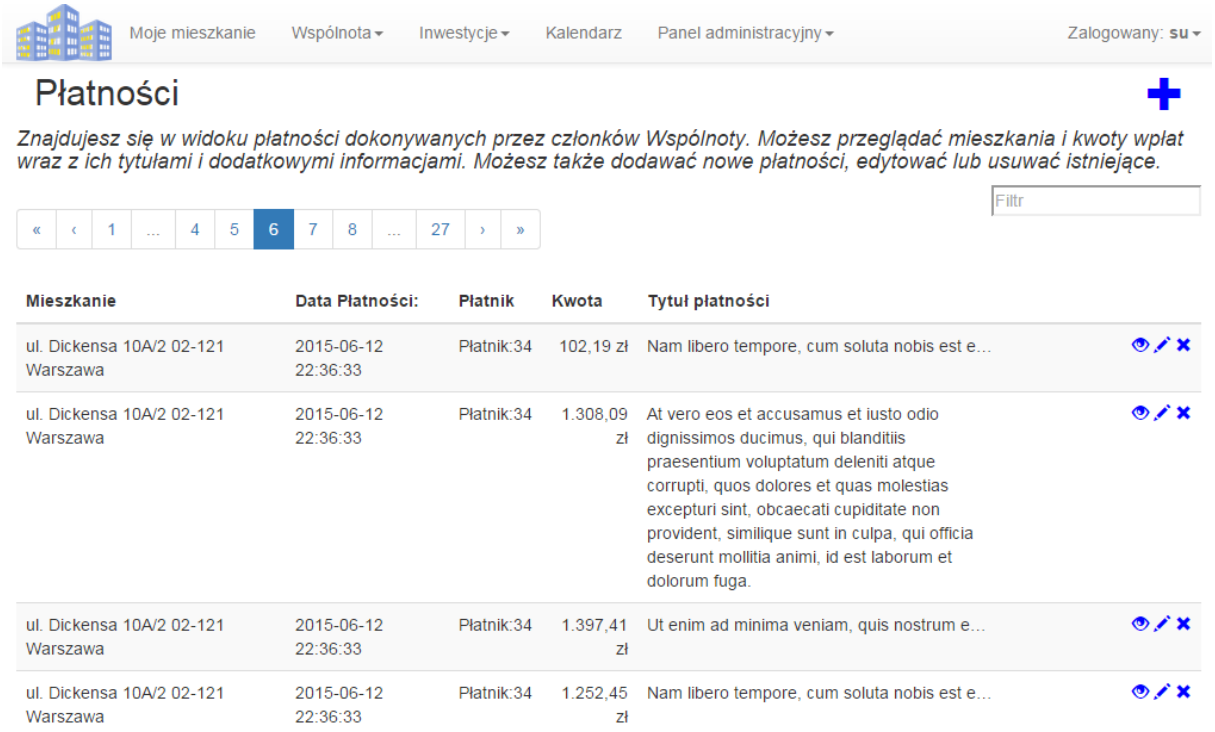
Do każdej uchwały możemy dodać załącznik który można pobrać po wejściu w szczegóły. Wygląd szczegółów przedstawia Rysunek 33.



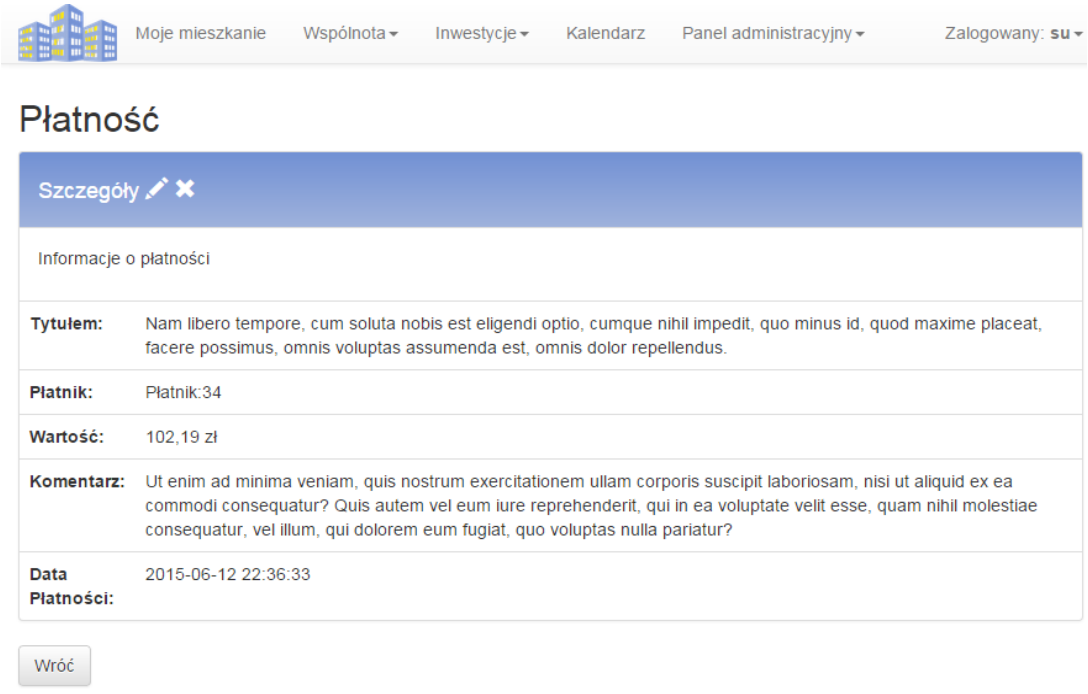
Rysunek 33. Szczegóły uchwały. Źródło: Opracowanie własne.

4.3.3 Płatności

W zakładce znajdziemy wszystkie płatności dokonane przez członków wspólnoty. Dane domyślnie zostały posortowane od najnowszych wpłat, zastosowano także stronicowanie. Po wejściu w szczegóły (Rysunek 35) wyświetla się komentarz do płatności. Kliknięcie w dowolny rekord powoduje ukazanie pełnej treści tytułu płatności (Rysunek 34).



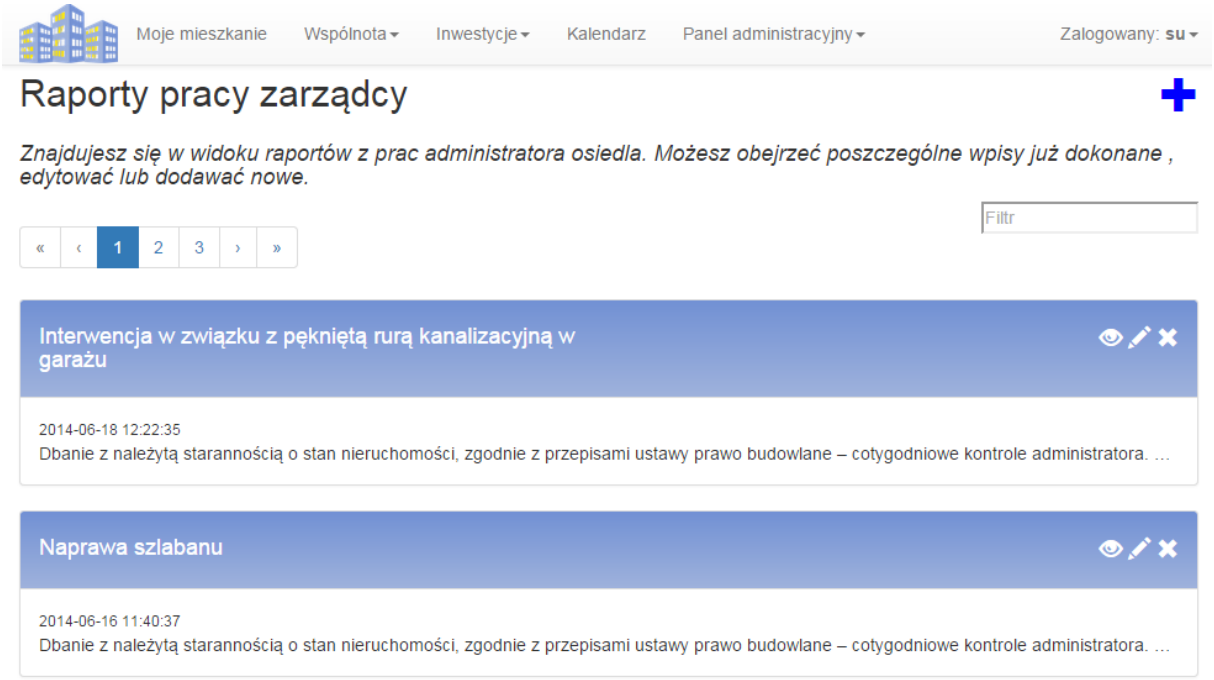
Rysunek 34. Widok płatności. ***Źródło: Opracowanie własne.***



Rysunek 35. Szczegóły płatności. ***Źródło: Opracowanie własne.***

4.3.4 Raporty

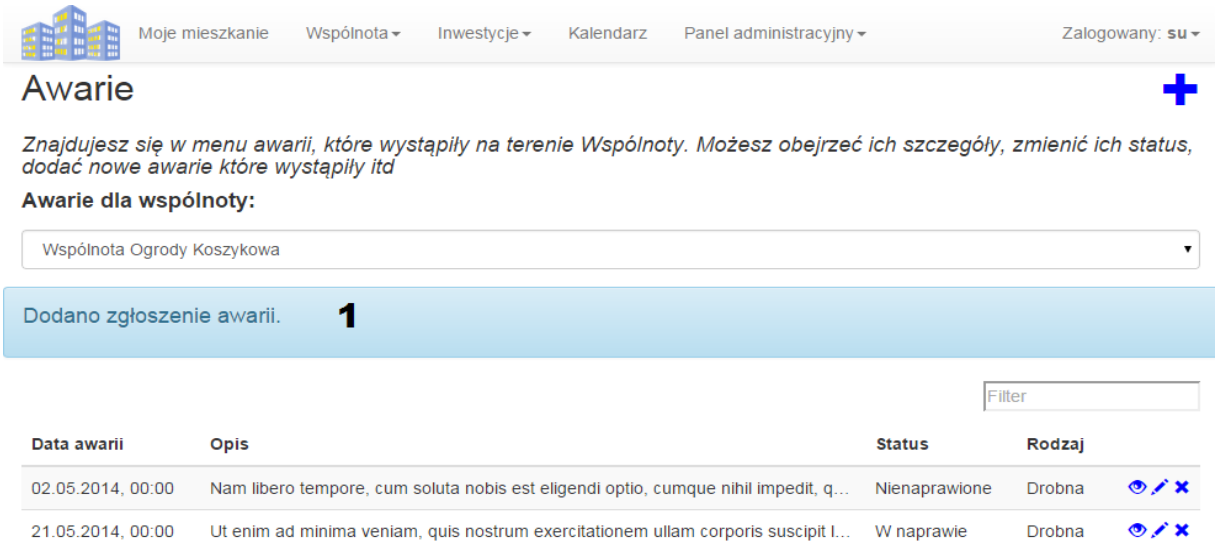
Widok raportów zarządcy ma służyć sprawozdaniu wszystkich prac, które zostały dokonane przez administratora w określonym czasie. Najnowsze wpisy są wyświetlane na samej górze, a zastosowanie zmodyfikowanej wersji „kafla" ma na celu uporządkowanie wyglądu i nawiązanie do wcześniejszych rozwiązań.



Rysunek 36. Raporty Zarządcy. *Źródło: Opracowanie własne.*

4.3.5 Awarie

Sposób reprezentowania awarii został rozwiązany w analogiczny sposób do płatności. Elementem o którym wcześniej nie wspominaliśmy, a występującym w całej aplikacji jest potwierdzenie dodania, usunięcia bądź edycji obiektu (pkt. 1 - Rysunek 31).

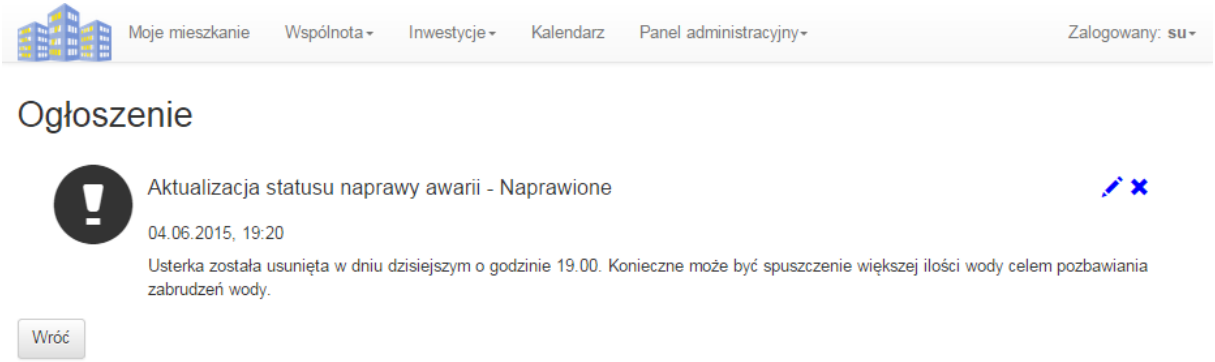


Rysunek 37. Widok awarii. *Źródło: Opracowanie własne.*

Dodatkowym elementem jest „Status awarii”, który po wejściu w edycję można zmieniać tak, aby mieszkaniec był na bieżąco informowany o postępie i zakresie prowadzonych prac. Nowo dodane awarie, bądź te w których został zmieniony status automatycznie tworzą ogłoszenie (Rysunek 38).

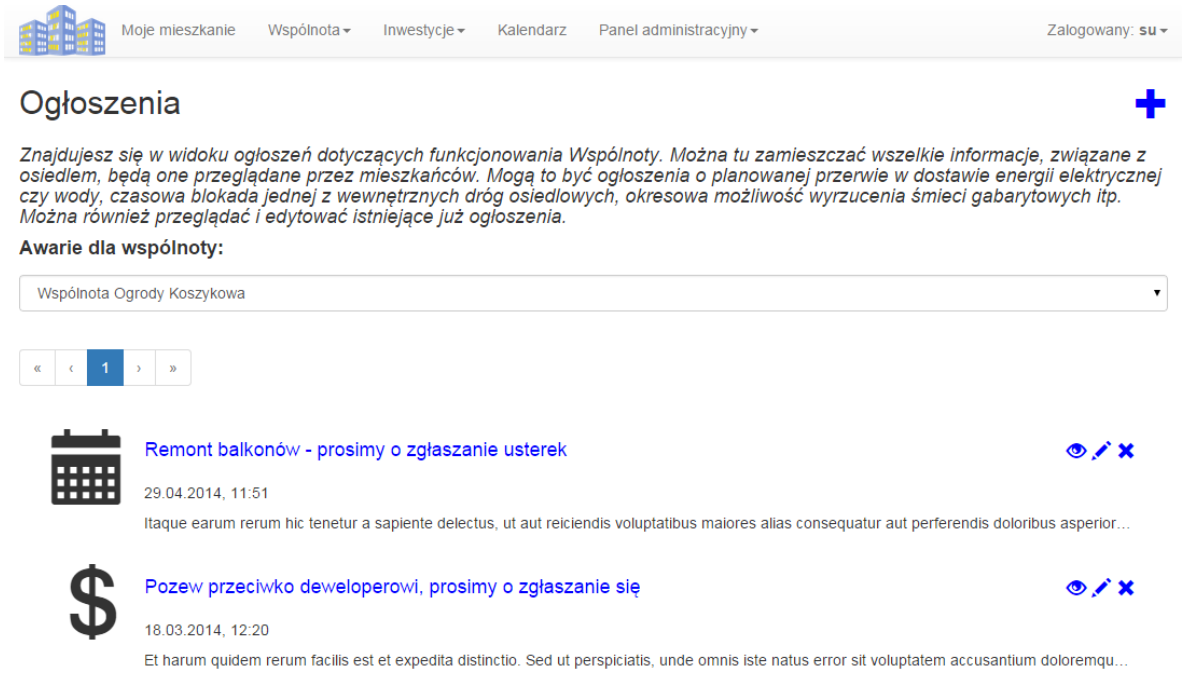
4.3.6 Ogłoszenia

Ogłoszenia są jednym z ważniejszych elementów projektowanego systemu. To dzięki nim użytkownik na bieżąco jest informowany o wydarzeniach, które mają miejsce w jego wspólnocie. Ogólny widok został ograniczony do 150 znaków treści ogłoszenia. Pełna zawartość zostaje przedstawiona po wejściu w szczegóły. Rysunek 38 prezentuje wygląd szczegółów ogłoszenia.



Rysunek 38. Szczegóły ogłoszeń. ***Źródło: Opracowanie własne.***

Tak jak zostało to wspomniane wcześniej, dostęp do ogłoszeń nie wymaga logowania (27) jednak po zalogowaniu można wybrać wspólnotę, z której ogłoszenia będą wyświetlane oraz dokonać edycji. (Rysunek 39).

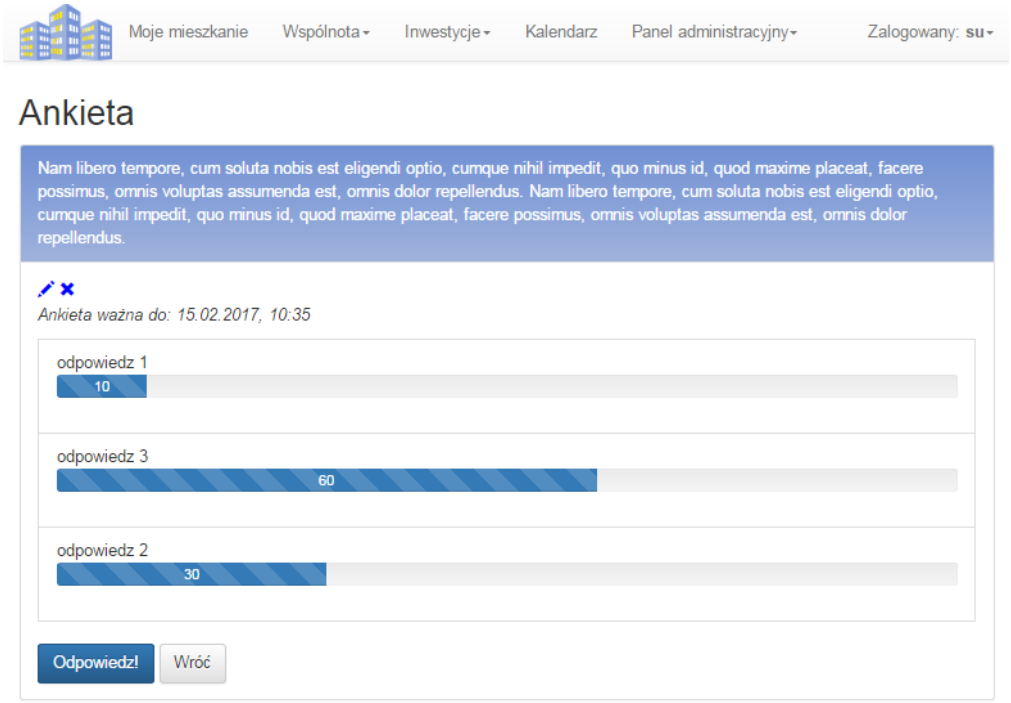


Rysunek 39. Widok ogłoszeń. ***Źródło: Opracowanie własne.***

4.3.7 Ankiety

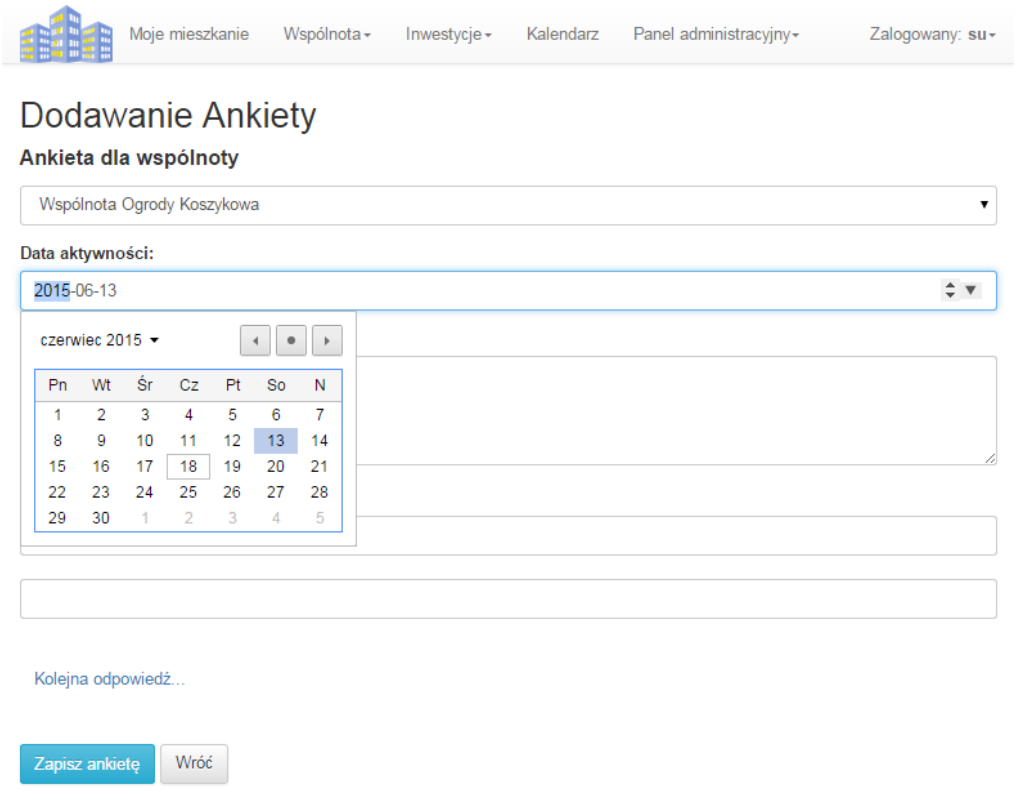
Elementem, którego próżno szukać w konkurencyjnych rozwiązaniach (3) są ankiety. Ich zastosowanie ma na celu zbieranie opinii na temat planowanych inwestycji bądź oceny prac zarządu.

W każdej ankiecie mamy możliwość dodania kilku opcji odpowiedzi (Rysunek 41). Aby oddać głos, należy wejść w szczegóły (Rysunek 40), wyboru możemy dokonać tylko jeden raz.



Rysunek 40. Szczegóły ankiety. ***Źródło: Opracowanie własne.***

Elementem, o którym wcześniej nie wspomniano, a niewątpliwie jest rozwiązaniem koniecznym, to wybór daty za pomocą widoku kalendarza.



Rysunek 41. Dodawanie ankiety. ***Źródło: Opracowanie własne.***

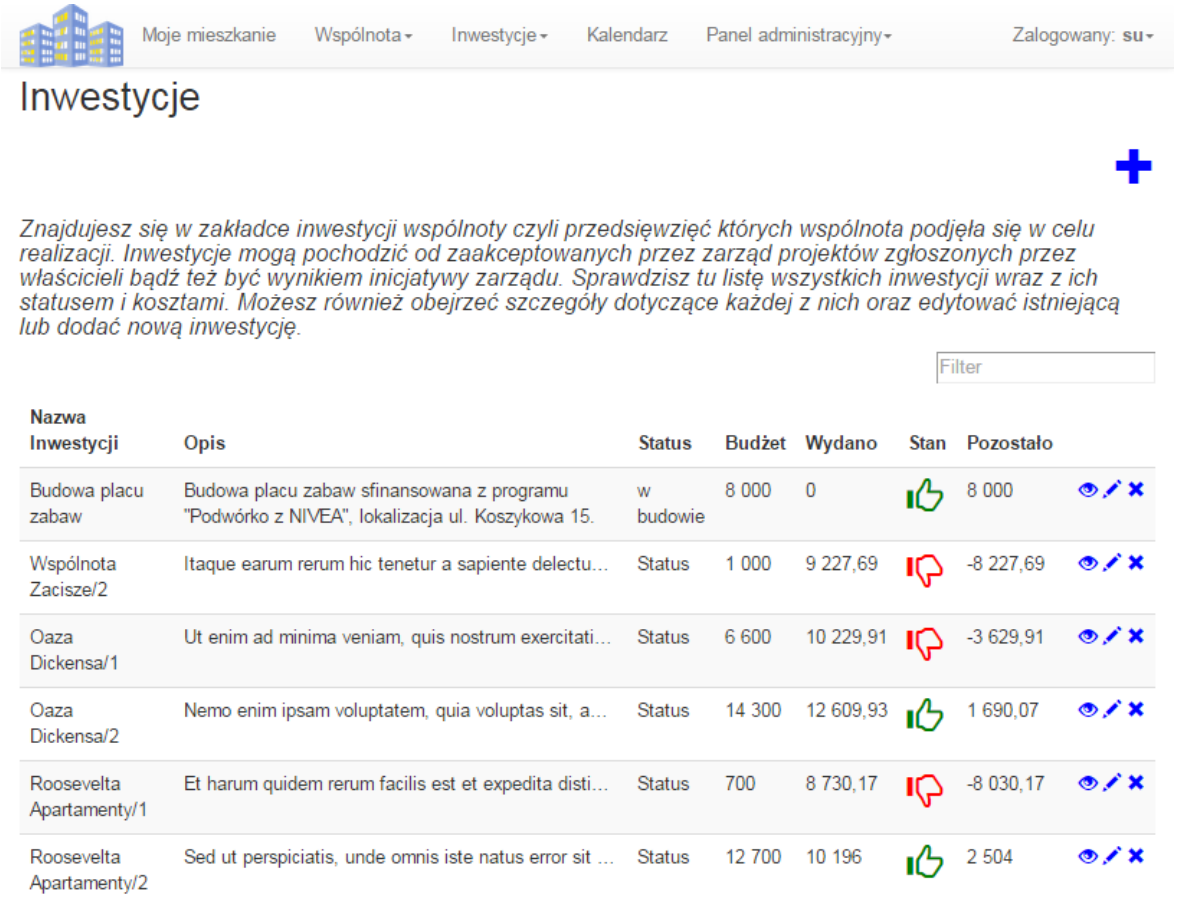
Takie rozwiązanie pojawia się wszędzie tam, gdzie zachodzi konieczność wprowadzenia daty. W przypadku ankiet data określa termin, do którego ankieta będzie aktywna dla użytkownika. Po tym czasie niemożliwe jest oddanie głosu.

4.4. Inwestycje

Zakładka inwestycje zawiera wszelakie informacje na temat prowadzonych inwestycji, projektów oraz operacji finansowych, dokonanych przez wspólnotę. Rozwiązanie to ma za zadanie przedstawiać na bieżąco stan finansowy wspólnoty oraz zakres prowadzonych prac. Celem jest uniknięcie przekroczenia planowanego budżetu wspólnoty.

4.4.1 Lista inwestycji

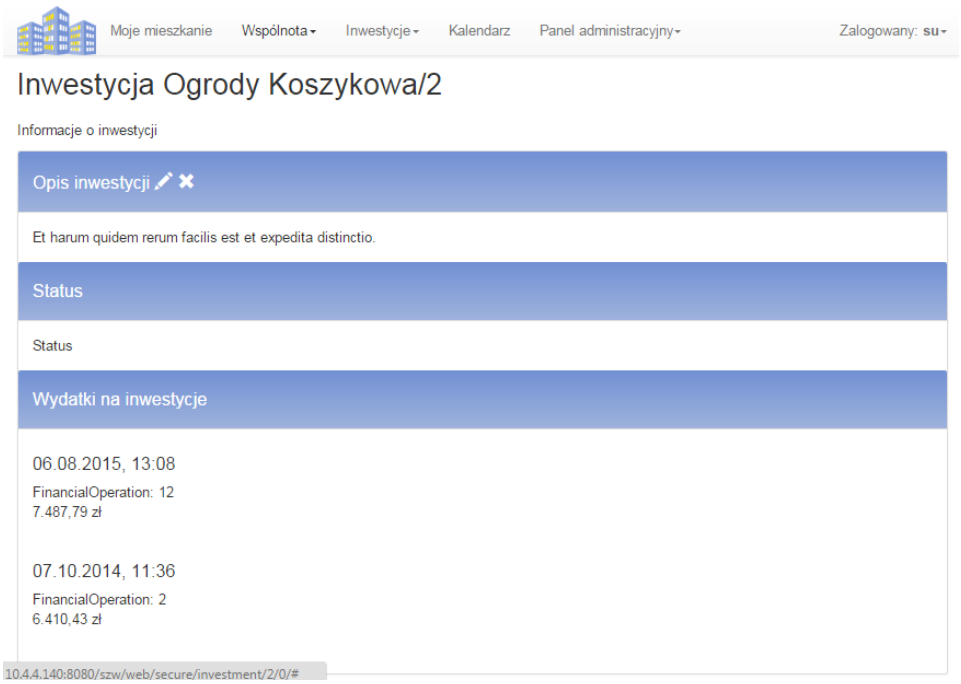
Lista została przedstawiona w analogiczny sposób do rozwiązania wyżej (4.3.2) - za pomocą tabeli, z filtrowaniem i ikonami akcji w każdym wierszu. Pełen opis dostępny jest po kliknięciu w dowolny rekord interesującej nas inwestycji. Status wpisujemy ręcznie, co zostało uzasadnione w przypadku biznesowym. Jako, iż żyjemy w świecie symboli, postanowiono stan inwestycji zaprezentować za pomocą popularnego „lajka". Jeżeli inwestycja przekroczy budżet z automatu, status ustawiany jest na czerwoną łapkę w dół (Rysunek 42).



Nazwa Inwestycji	Opis	Status	Budżet	Wydano	Stan	Pozostało	
Budowa placu zabaw	Budowa placu zabaw sfinansowana z programu "Podwórko z NIVEA", lokalizacja ul. Koszykowa 15.	w budowie	8 000	0		8 000	
Wspólnota Zacisze/2	Itaque earum rerum hic tenetur a sapiente delectu...	Status	1 000	9 227,69		-8 227,69	
Oaza Dickensa/1	Ut enim ad minima veniam, quis nostrum exercitati...	Status	6 600	10 229,91		-3 629,91	
Oaza Dickensa/2	Nemo enim ipsam voluptatem, quia voluptas sit, a...	Status	14 300	12 609,93		1 690,07	
Roosevelta Apartamenty/1	Et harum quidem rerum facilis est et expedita disti...	Status	700	8 730,17		-8 030,17	
Roosevelta Apartamenty/2	Sed ut perspiciatis, unde omnis iste natus error sit ...	Status	12 700	10 196		2 504	

Rysunek 42. Widok inwestycji. ***Źródło: Opracowanie własne.***

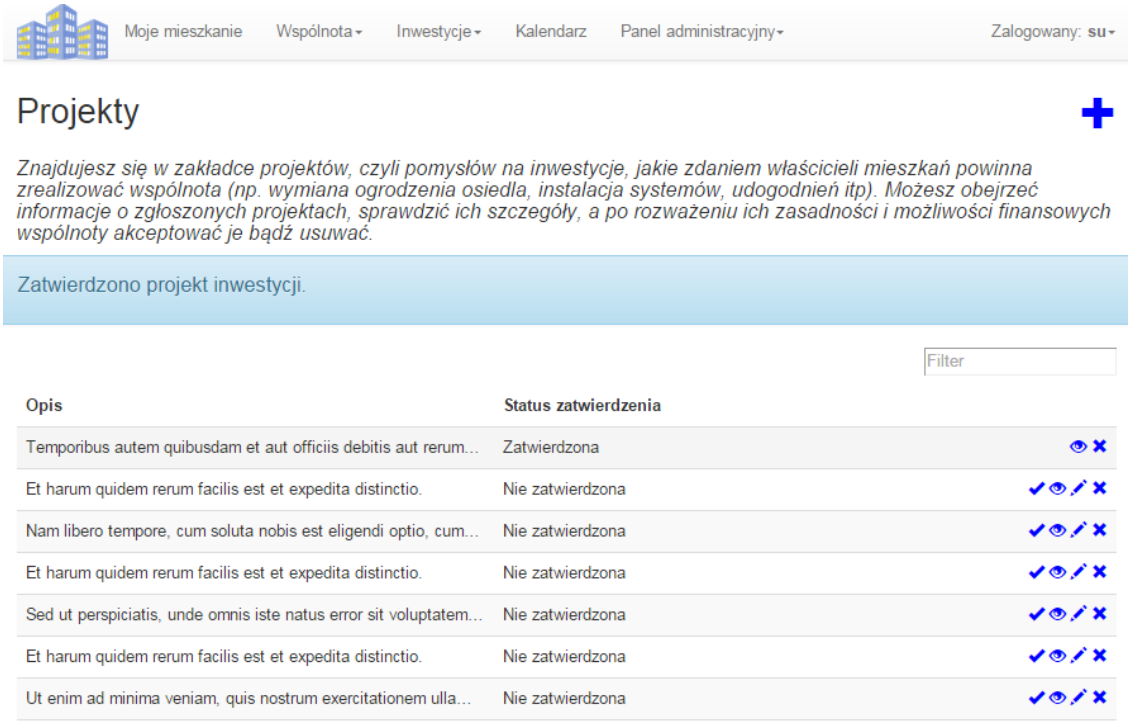
Po wejściu w szczegóły inwestycji (Rysunek 43) wyświetlana jest historia operacji finansowych. Dzięki takiemu rozwiązaniu na bieżąco można śledzić wszystkie wpłaty, które dokonano na poczet prac oraz kontrolować zaplanowany budżet.



Rysunek 43. Szczegóły inwestycji. ***Źródło: Opracowanie własne.***

4.4.2 Projekty

Zakładka projektów służy do zgłaszania propozycji inwestycji przez właścicieli mieszkań, które według ich zdania, wspólnota powinna zrealizować. Aby zaakceptować projekt, dodano ikonę akceptacji. Gdy projekt zostanie zatwierdzony, automatycznie zostaje przeniesiony do inwestycji, gdzie czeka na realizację.



Rysunek 44. Projekty inwestycji. ***Źródło: Opracowanie własne.***

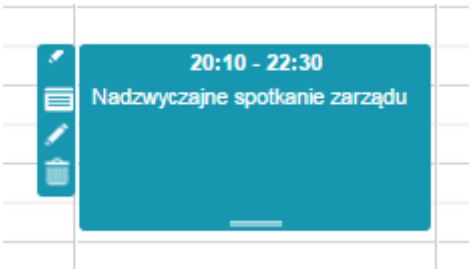
4.4.3 Operacje finansowe

Widok operacji finansowych przedstawia pełną listę wpłat dokonanych przez wspólnotę. Operacje tyczą się tylko i wyłącznie inwestycji znajdujących się w liście (4.4.1). W celu poprawy czytelności danych, zastosowano wyświetlanie po wspólnocie. Podlinkowana została również nazwa inwestycji która przenosi nas bezpośrednio w szczegóły.

Rysunek 45. Operacje finansowe. ***Źródło: Opracowanie własne.***

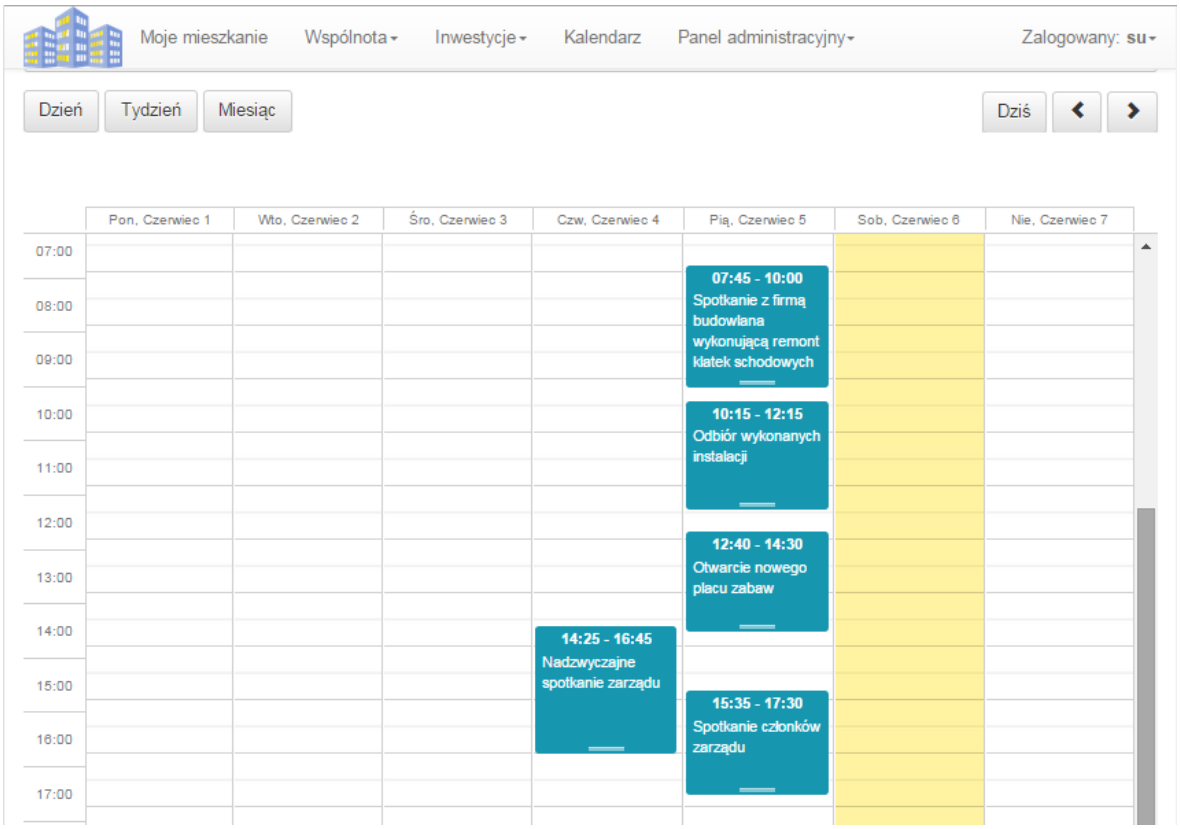
4.5. Kalendarz

Kalendarz (Rysunek 47) jest uporządkowanym spisem dni, tygodni i miesięcy, służący do odmierzania czasu. Jest jednym z wielu ważnych elementów, który ułatwia planowanie i organizacje prac zarządu. Dzięki niemu, na bieżąco można sprawdzać wpisane w nim wydarzenia, dodawać nowe oraz edytować stare.



Rysunek 46. Wydarzenie w kalendarzu. ***Źródło: Opracowanie własne.***

Jego obsługa jest niezwykle intuicyjna, wystarczy kliknąć dwukrotnie w interesujący nas dzień i przedział czasowy, aby dodać nowe wydarzenie. Rysunek 46 przedstawia widok wydarzenia w kalendarzu z ikonami akcji.



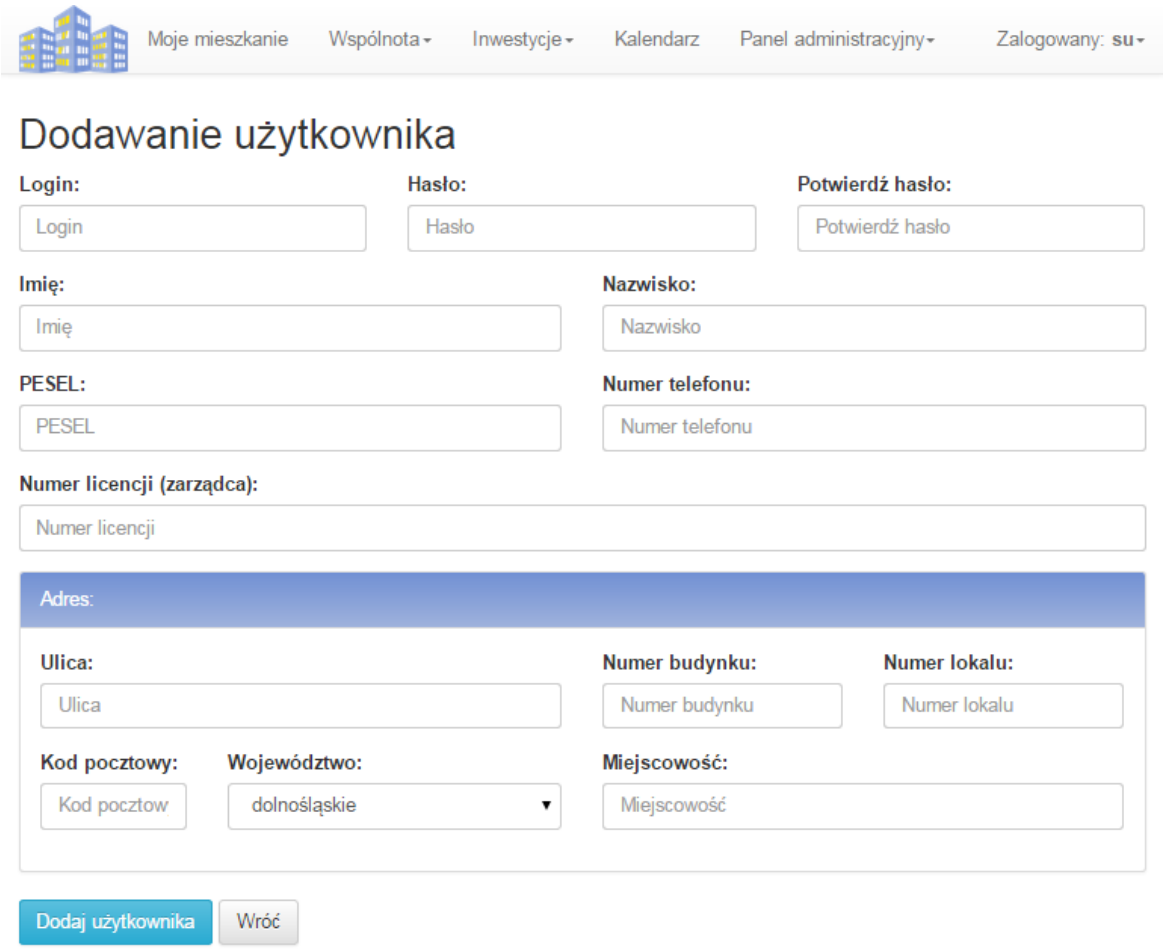
Rysunek 47. Kalendarz zarządu. *Źródło: Opracowanie własne.*

4.6. Panel administracyjny

Aby sprawnie zarządzać wspólnotami w projektowanym systemie, konieczne było wykonanie panelu administracyjnego, który pozwoli na zarządzanie użytkownikami oraz obiektami budowlanymi. Wszystkie elementy zostaną przedstawione i opisane w poniższych punktach.

4.6.1 Zarządzenia użytkownikami

Nawet najlepiej zaprojektowany system bez sprawnie działającego modelu uprawnień jest bezwartościowy. Jego podstawą jest dodawanie użytkowników systemu. Wszystkie dane, które są konieczne przy rejestracji, zostają wprowadzone przez administratora systemu, przedstawia Rysunek 48.



Rysunek 48. Formularz rejestracji użytkownika. *Źródło: Opracowanie własne.*

Aby nadać jakiekolwiek uprawnienia, musimy wejść w szczegóły użytkownika, gdzie znajdują się podstawowe dane (Rysunek 49) wpisane w formularzu rejestracji oraz tabele uprawnień (Rysunek 50).

Rysunek 49. Dane użytkownika. *Źródło: Opracowanie własne.*

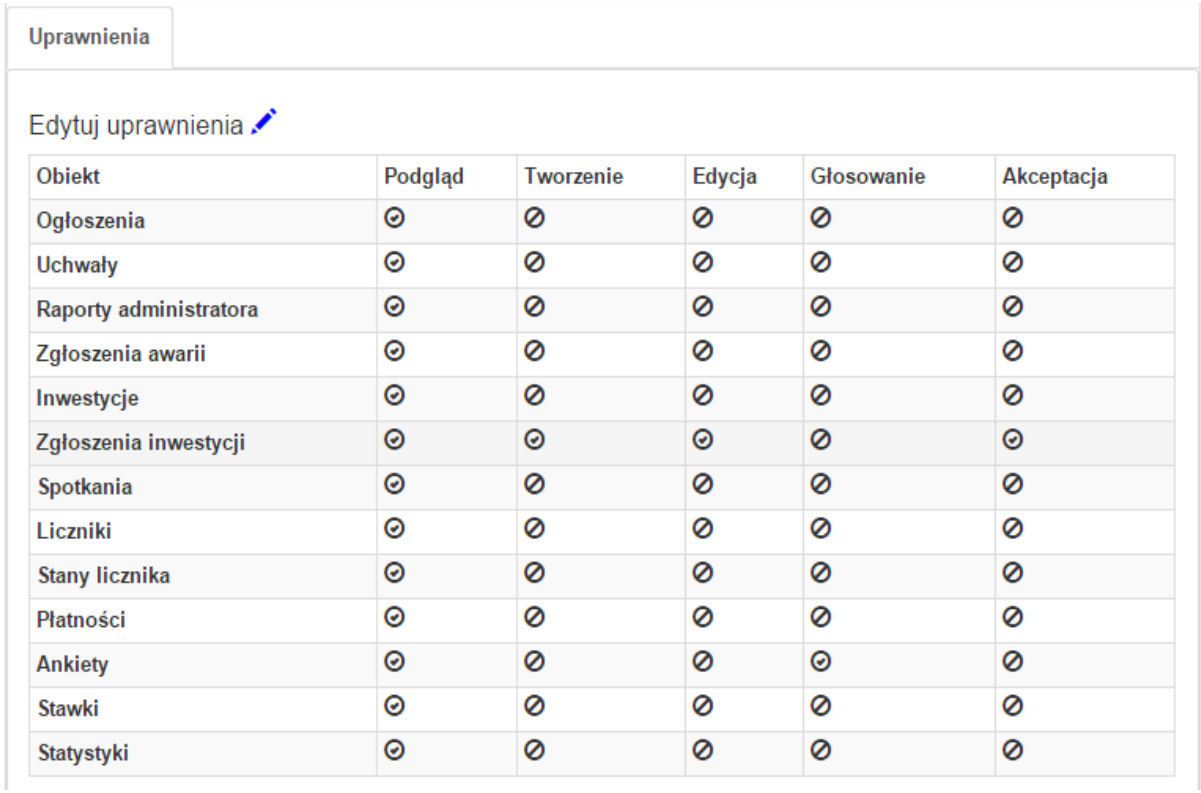
W zaprojektowanym systemie uprawnienia możemy podzielić na dwa rodzaje:

- biznesowe,
- systemowe.

Za pomocą uprawnień biznesowych projektowanego systemu możemy nadać użytkownikowi rolę administratora systemu, właściciela, lokatora, księgowości, zarządcy, członka zarządu i powiązać go z konkretną wspólnotą, w której będzie mógł wykonywać wszystkie operacje wynikające z uprawnień ról.

Rysunek 50. Tabela uprawnień. *Źródło: Opracowanie własne.*

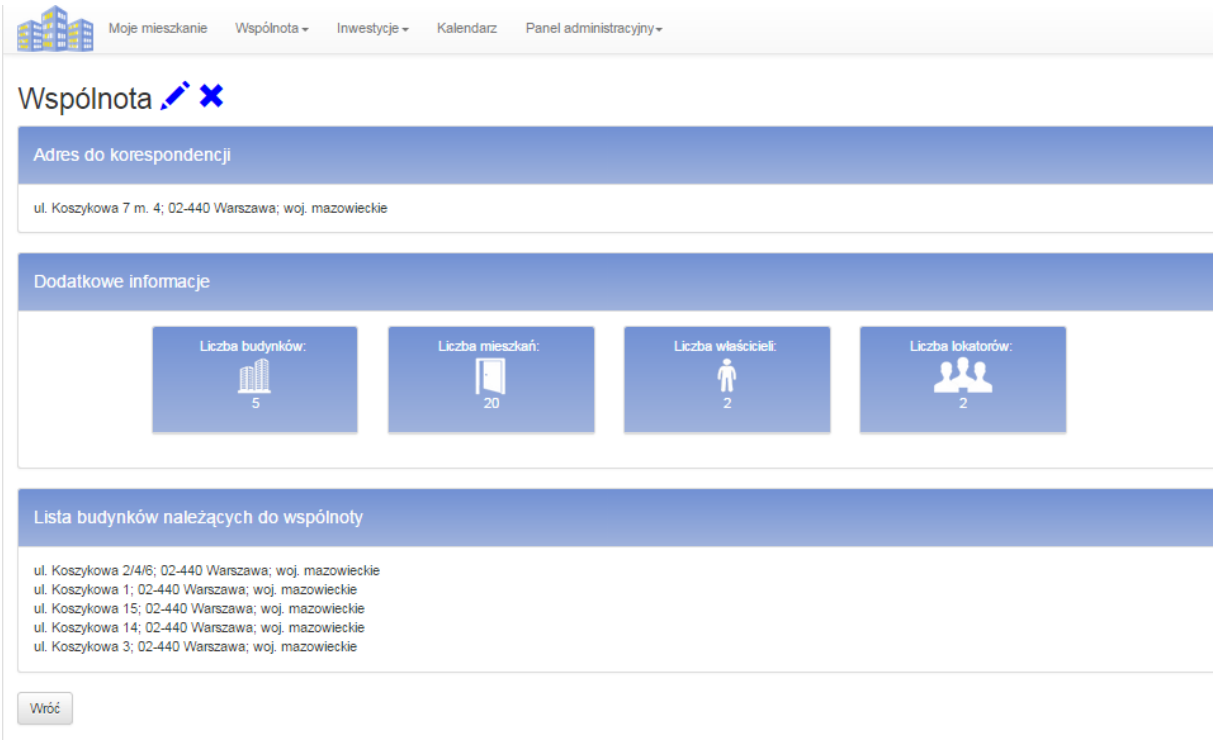
Prawa dostępu i podstawowych akcji użytkownika do danych obiektów zaprezentowano za pomocą osobnej tabeli przedstawionej na Rysunek 51. Aby przejść do ich zmiany, wystarczy kliknąć w ikonę edycji.



Rysunek 51. Uprawnienia do obiektów. *Źródło: Opracowanie własne.*

4.6.2 Wspólnoty

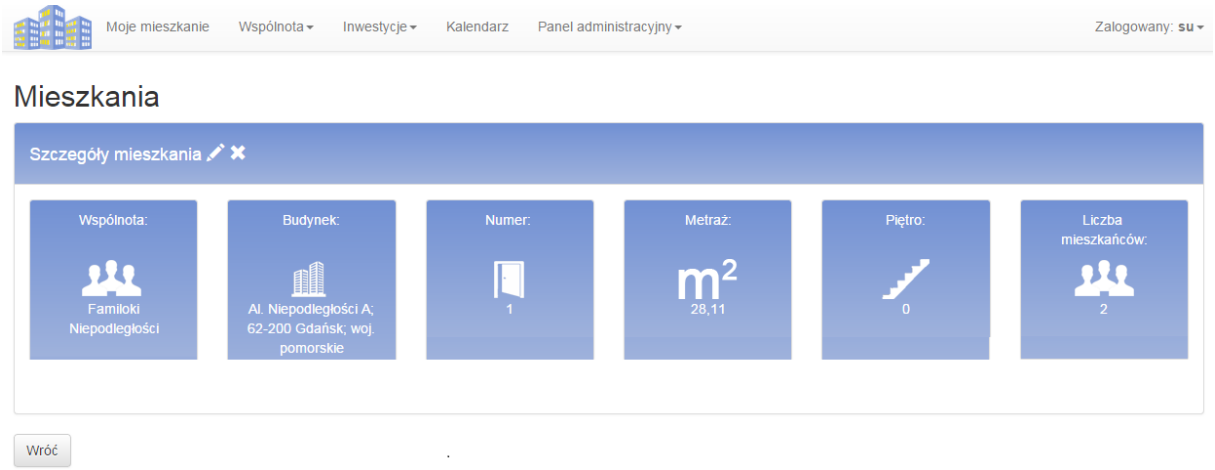
Widok wspólnot wyświetlany jest za pomocą tabeli z podstawowymi ikonami akcji, gdzie można dodawać, edytować bądź usuwać istniejące elementy. Po wejściu w szczegóły (Rysunek 52), wyświetlany są adres korespondencyjny oraz podstawowe dane na temat liczby budynków, mieszkań oraz lokatorów. Znajdziemy tu także listę budynków należących do wspólnoty wraz z ich adresami.



Rysunek 52. Szczegóły wspólnoty. ***Źródło: Opracowanie własne.***

4.6.3 Mieszkania

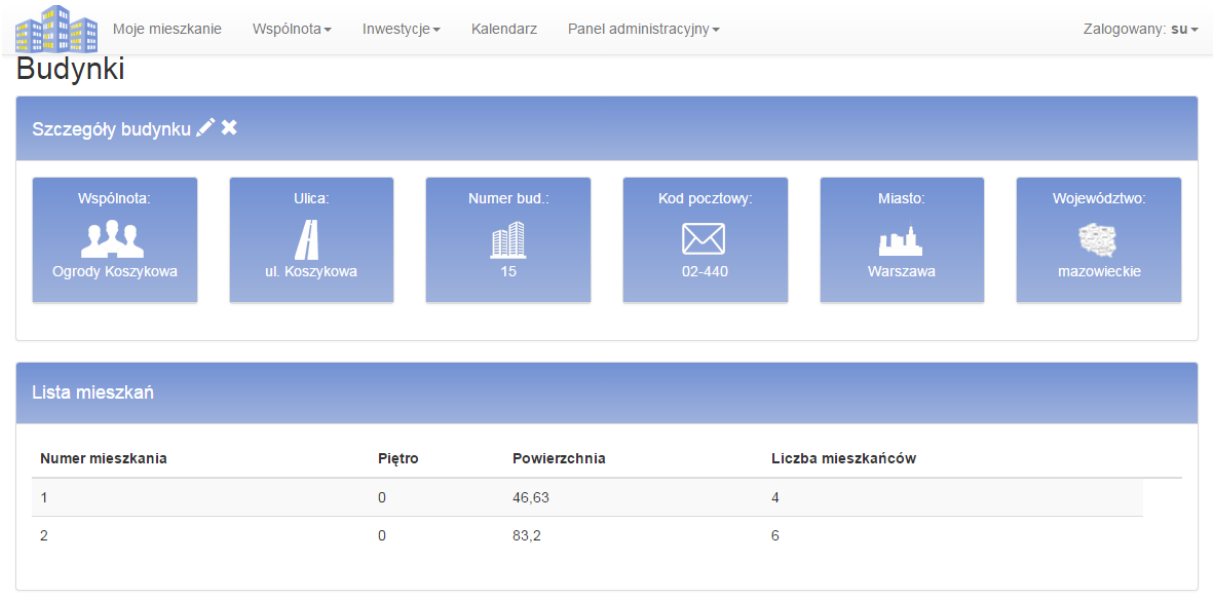
W zakładce mieszkań znajdują się podstawowe informacje na temat mieszkania oraz przypis, do której z wspólnot mieszkanie należy.



Rysunek 53. Widok mieszkania. ***Źródło: Opracowanie własne.***

4.6.4 Budynki

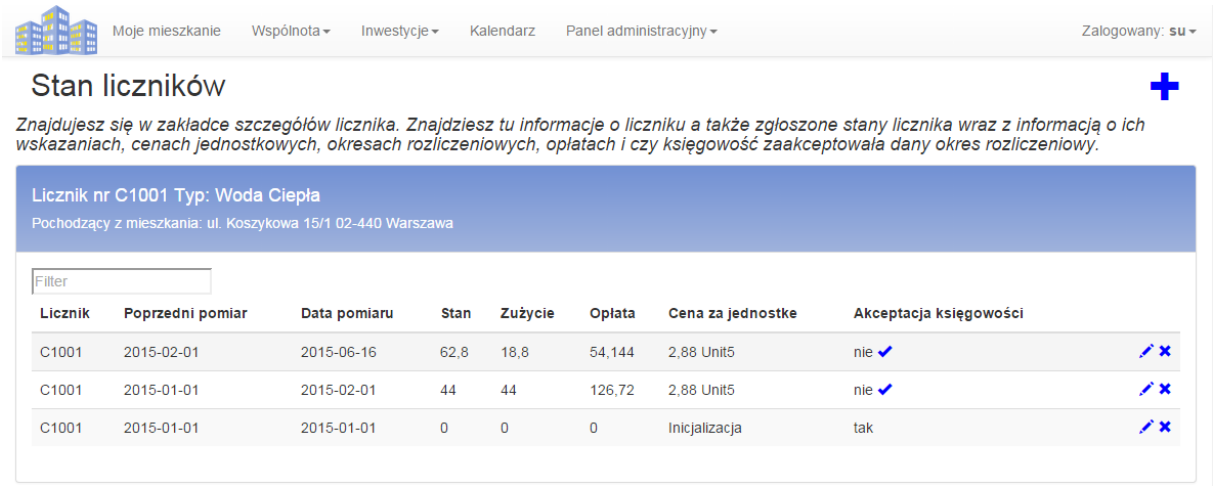
Sposób wyświetlania informacji o budynkach jest zrealizowany w analogiczny sposób do widoku wspólnoty. W celu identyfikacji dodano informacje, do której z wspólnot budynek należy. Oprócz danych adresowych znajdziemy tu także listę mieszkań w budynku wraz z ich numerem, metrażem oraz liczbą osób je zamieszkujących. Rozwiązanie przedstawia Rysunek 54.



Rysunek 54. Szczegóły budynku. *Źródło: Opracowanie własne.*

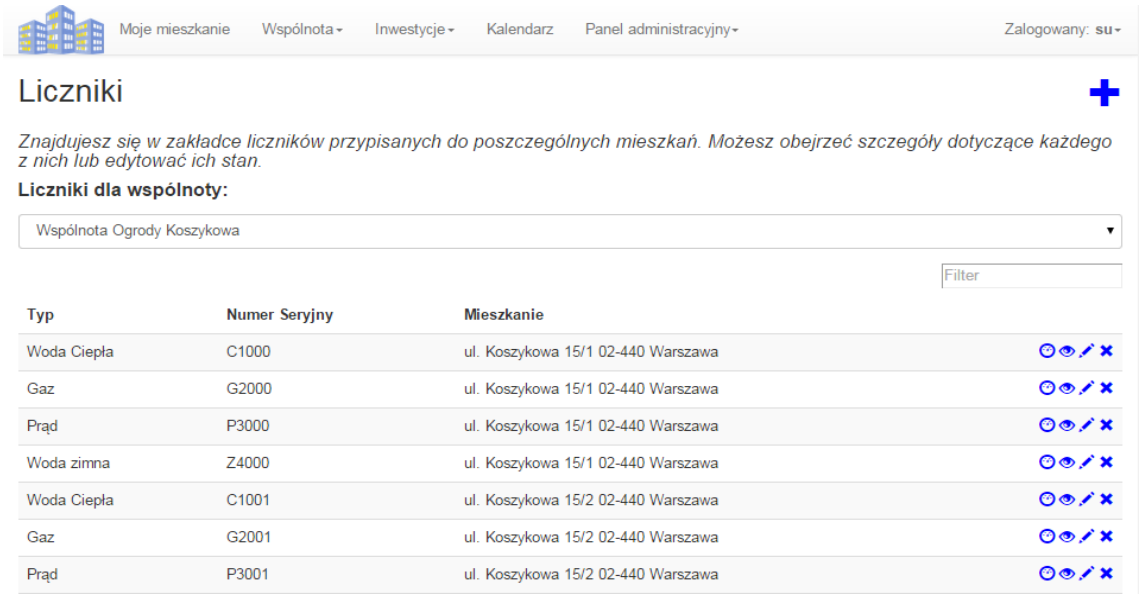
4.6.5 Liczniki

W zakładce znajdziemy wszystkie typy liczników (Rysunek 56) wraz z ich numerem seryjnym i przypisem do mieszkania w danej wspólnocie. Po wejściu w szczegóły (Rysunek 55) ukazywany jest stan licznika wraz z zużyciem za dany okres pomiędzy pomiarami.



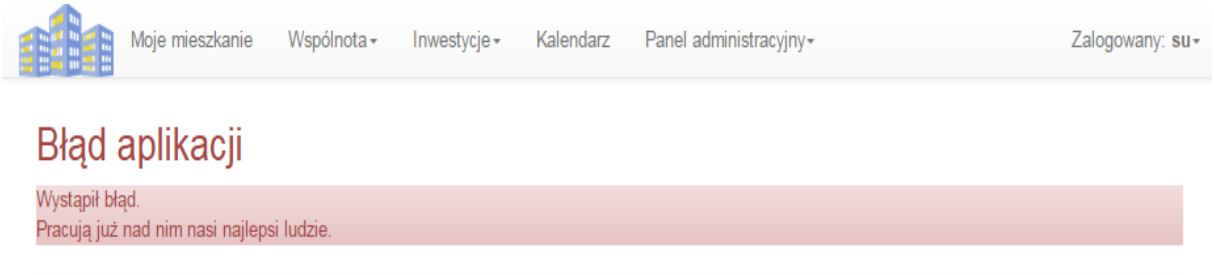
Rysunek 55. Stan licznika. *Źródło: Opracowanie własne.*

Zużycie jest automatycznie mnożone przez jednostkę, co zwraca nam wysokość opłaty za dany okres rozliczeniowy. Każdy stan musi być zaakceptowany przez księgowość.



Rysunek 56. Widok liczników. ***Źródło: Opracowanie własne.***

Podsumowując, system zarządzania wspólnotami jest prototypem, który wymaga dalszych prac choćby po to , aby pozbawić go problemów „wieku dziecięcego". Przedstawione rozwiązanie jest propozycją systemu, który według nas, osób go tworzących, jest dobrą podstawą do kontynuowania rozwoju i poszerzania funkcjonalności, tak aby SZW stał się aplikacją konkurencyjną na rynku. Tak jak wspomniano, jest to prototyp, a więc w niektórych momentach nie bądźmy zaskoczeni widokiem przedstawionym na poniższym rysunku (Rysunek 57).



Rysunek 57. Błąd aplikacji. ***Źródło: Opracowanie własne.***

5. Wykorzystane technologie

W rozdziale tym zostaną przedstawione wszystkie ważniejsze technologie wykorzystane do stworzenia „Systemu zarządzania wspólnotami mieszkaniowymi”.

5.1. Java 8

Java 8 udostępniona przez firmę Oracle to najświeższa wersja oprogramowania, która zawiera najnowsze funkcje, rozszerzenia oraz poprawki. Zadaniem ich jest zwiększenie wydajności programowania oraz obsługi programów. Technologia ta została wykorzystana w niniejszej pracy.

5.1.1 Wstęp

Ten zaprojektowany przez Sun Microsystems (szerzej: w części poświęconej rysowi historycznemu) wysokopoziomowy, współbieżny, oparty na klasach obiektowy język programowania jest jednym z najpopularniejszych języków, którymi posługują się twórcy oprogramowania na całym świecie. Według [2] korzysta z niego 9 milionów twórców oprogramowania. Poniżej (Tabela 2) przedstawione zostały wyniki badań popularności poszczególnych języków wg TIOBE Index [3].

Tabela 2. Popularność poszczególnych Języków. *Źródło: TIOBE Software* [2].

mar-15	mar-14	Zmiana	Język Programo - wania	Wskaźnik	Zmiana
1	1		C	16.642%	-0.89%
2	2		Java	15.580%	-0.83%
3	3		Objective-C	6.688%	-5.45%
4	4		C++	6.636%	+0.32%
5	5		C#	4.923%	-0.65%
6	6		PHP	3.997%	+0.30%
7	9	↑	JavaScript	3.629%	+1.73%
8	8		Python	2.614%	+0.59%
9	10	↑	Visual Basic .NET	2.326%	+0.46%
10	-	⇈	Visual Basic	1.949%	+1.95%
11	12	↑	F#	1.510%	+0.29%
12	13	↑	Perl	1.332%	+0.18%
13	15	↑	Delphi/Object Pascal	1.154%	+0.27%
14	11	↓	Transact-SQL	1.149%	-0.33%
15	21	⇈	Pascal	1.092%	+0.41%
16	31	⇈	ABAP	1.080%	+0.70%
17	19	↑	PL/SQL	1.032%	+0.32%
18	14	⇊	Ruby	1.030%	+0.06%
19	20	↑	MATLAB	0.998%	+0.31%
20	45	⇈	R	0.951%	+0.72%

Język Java w swoich założeniach, jako język wysokiego poziomu, ma być przyjaznym użytkownikowi narzędziem tworzenia aplikacji, które – opierając się na stworzonym przez Sun Microsystems sloganie „Write once, run anywhere” (WORA)[23] – raz napisane i skompilowane mogą

zostać uruchomione na wszystkich wspierających Java urządzeniach, bez konieczności ponownej kompilacji. Funkcjonalność powyższa wynika z faktu, że aplikacje stworzone w języku Java podlegają kompilacji do kodu bitowego, który z kolei może zostać uruchomiony na każdej maszynie wirtualnej Javy (*Java Virtual Machine JVM*), całkowicie niezależnie od rodzaju architektury danego komputera.

5.1.2 Historia

We wczesnych latach 90. zwiększenie znaczenia komputerowych rozwiązań sieciowych dostępnych w życiu codziennym stanowiło ambitne wyzwanie. W roku 1991 niewielka grupa inżynierów zatrudnionych w firmie Sun, nazywających siebie „Green Team", przyjęła śmiałe wówczas założenie, że na rynku komputerowym pojawi się fala łączenia cyfrowych urządzeń przeznaczonych dla konsumentów z komputerami. Grupa pod kierownictwem Jamesa Goslinga, pracując wytrwale, stworzyła język programowania, który miał zrewolucjonizować świat – Javę [5]. Miał on umożliwić sprawną obsługę urządzeń cyfrowych, takich jak chociażby dekodery czy telewizory. Początkowo rezultat prac Green Teamu otrzymał nazwę „Greentalk" [6], nadaną przez Jamesa Goslinga, a rozszerzeniem pliku było .gt. Wkrótce język nazwany został mianem „Oak". Do tej pory dominującym językiem programowania był C++. Java miała w swoim założeniu oferować użyteczne funkcjonalności tego języka przy jednoczesnej eliminacji tych bardziej uciążliwych [7].

W roku 1995 Oak został przemianowany na znaną dziś Javę [7], jako że stanowił już znak handlowy Oak Technologies. Interesującym wydaje się być wątek nadania projektowi nazwy Java. Otóż zespół, mając stworzyć nową nazwę, rozważał kilka możliwości. Rozważano m.in. takie nazwy, jak „dynamic", „revolutionary", „silk", „jolt", czy „DNA". Poszukiwano nazwy, która oddałaby istotę nowej technologii – rewolucyjnej, dynamicznej, energicznej, fajnej, unikalnej, łatwej do przeliterowania i wymówienia. Jak przyznał później James Gosling, Java i Silk były najlepszymi wyborami i między nimi rozegrał się końcowy pojedynek. Ostatecznie większość członków zespołu opowiedziała się za Javą.

W 1995 r. ogłoszono, że przeglądarka internetowa Netscape Navigator ma być oparta właśnie na Javie. Zapowiedzieli to 23 maja 1995 r. John Gage, dyrektor Biura Naukowego w Sun Microsystems, i Marc Anderseen, współzałożyciel i wiceprezes zarządu Netscape.

5.1.3 Wersje Java

Java SE 8 jest najnowszą stabilną edycją, do której od momentu jej wydania do 3 marca 2015 r. udostępniono sześć znaczących aktualizacji.

- JDK Alpha and Beta (1995 r.)
- JDK 1.0 (23 stycznia 1996 r.)
- JDK 1.1 (19 lutego 1997 r.)
- J2SE 1.2 (8 grudnia 1998 r.)
- J2SE 1.3 (8 maja 2000 r.)
- J2SE 1.4 (6 lutego 2002 r.)
- J2SE 5.0 (30 września 2004 r.)
- Java SE 6 (11 grudnia 2006 r.)
- Java SE 7 (28 lipca 2011 r.)
- Java SE 8 (18 marca 2014 r.)

[3] „Napisz raz, uruchom gdziekolwiek" (tłum. własne) – http://www.computerweekly.com/feature/Write-once-run-anywhere , występuje również w formie „Write once, run everywhere" (WORE).

5.1.4 Wybór Javy

Jako główne, najczęściej wymieniane języki mogące być alternatywą dla Javy należałoby z pewnością wymienić C++, który stanowił inspirację dla twórców Javy, oraz C#, który inspirowany był językiem Java. *James Gosling stwierdził nawet, że C# stanowi imitację Javy, dodając, że jest to coś na kształt Javy, z tym że pozbawione niezawodności, wydajności i bezpieczeństwa* [9]. Wśród pozostałych języków wysokiego poziomu wskazać można by chociażby Python, Ruby czy PHP.

Oczywiście można postawić zasadnicze pytanie, dlaczego, wobec takiej liczby dostępnych języków programowania, na potrzeby niniejszej pracy wybrana została akurat Java. Odpowiedzią mogą być między innymi cechy tego języka, jak też jego popularność, która skutkuje również licznymi rozwiązaniami dającymi możliwość tworzenia kompleksowych, wewnętrznie spójnych rozwiązań programistycznych. Nie bez znaczenia jest również fakt, który był jednym z założeń twórców tego języka, że stworzone przez nich dzieło ma umożliwiać skrócenie i ułatwienie pracy programisty. Oczywiście, jak się okazało, kosztem wydajności samego programu, ale praca programisty została znacznie przyspieszona w stosunku do analogicznej, wykonanej np. w C++.

Wymienić tu można również obsługę wielu wątków, programowania sieciowego czy biblioteki [9]. To tylko niektóre z powodów, dla których na potrzeby niniejszej pracy inżynierskiej została wybrana Java.

Podsumowując – czym jest Java i dlaczego została wybrana:

- Jest to język prosty, oczyszczony z rodzących problemy elementów składni, jakimi cechuje się chociażby C++.
- Charakteryzuje się obiektowością oraz zastąpieniem wielodziedziczenia prostszymi interfejsami.
- Jest to język sieciowy, wyposażony w bogatą bibliotekę procedur wspomagających programowanie sieciowe, w tym współpracę z protokołami takimi, jak np. TCP/IP, HTTP czy FTP.
- Charakteryzuje się względną niezawodnością – jej model jest zaprojektowany w sposób uniemożliwiający zniszczenie danych poprzez nadpisanie pamięci.
- Zapewnia bezpieczeństwo w sieci poprzez tworzenie systemów odpornych na infekcje i ingerencję.
- Jest językiem niezależnym od architektury sprzętu. Kod podlega łatwej interpretacji na każdym urządzeniu i jest tłumaczony na kod maszynowy „w locie”.
- Jest językiem przenośnym, niezależnym od implementacji – w przeciwieństwie do C/C++. Rozmiary podstawowych typów i działania na nich są określone.
- Jest językiem interpretowalnym i wysoko wydajnym dzięki tłumaczeniu w locie na kod maszynowy.
- Pozwala na wielowątkowość w czasie rzeczywistym.
- Jest językiem dynamicznym, dostosowującym się do ewoluującego środowiska [9].

5.1.5 Frameworki Javy

Frameworki Java to zestaw bibliotek zapewniających interfejs użytkownika czy też „warstwę widoku” dla webowych aplikacji opartych na języku Java. Służą one głównie definiowaniu stron internetowych i obsłudze generowanych przez nie żądań HTTP [12]. Oferują one wiele użyteczności, redukując czas pracy twórców aplikacji. Frameworki skracają czas pisania kodu przede wszystkim dzięki wbudowanym w nie modułom i funkcjom. W kolejnych punktach zostaną omówione najpopularniejsze z nich, użyte w naszym projekcie.

5.2. WildFly

Poniżej przedstawiony zostanie serwer aplikacji WildFly, w przeszłości znany jako JBoss AS bądź po prostu Jboss. WildFly jest obecnie uznawany za jeden z najbardziej popularnych serwerów aplikacji. Dzięki niemu możemy wprowadzać dość rozbudowane i skomplikowane systemy, które korzystają z wielu rodzajów usług.

5.2.1 Wstęp

WildFly jest napisany całkowicie w języku programowania Java, dzięki czemu jego wielką zaletą jest dostępność multiplatformowa. Został on zbudowany na bazie JBoss Microcontainera, który przyjmuje rolę kontenera i pomaga bezpośrednio przy zarządzaniu cyklem życia obiektów POJO (Plain *Old Java Objects*), a także przy konfiguracji i instalacji aplikacji. W porównaniu do swoich poprzednich wersji, w tej odsłonie został znacząco zoptymalizowany proces rozruchowy – uruchamianie jednoczesne. WildFly uzupełniono również o konstrukcję modułową, zwiększono wykorzystywanie procesorów wielordzeniowych oraz polepszono zarządzanie pamięcią poprzez użycie „garbage collector".

5.2.2 Historia

Przedstawiając historie WildFly, musimy cofnąć się aż do 1999 roku, kiedy to pracę nad Jboss zaczął Marc Fleury. Produkt od samego początku był na licencji otwartej oraz nosił nazwę EJBoss (*Enterprise Java Beans Open Source Software*), pojawił się jednak sprzeciw ze strony firmy Sun, która ma zastrzeżone prawo do używania nazwy EJB. Postanowiono zatem porzucić E i produkt otrzymał nazwę Jboss. Kolejna, czyli już trzecia, wersja serwera została wydana w 2002 roku. Mogła się ona poszczycić absolutnym i całkowitym wsparciem specyfikacji J2EE, a ponadto uznano ją za pierwszą wersję, która mogła się mierzyć jak równy z równym z innymi komercyjnymi serwerami aplikacji, takimi jak np. WebSpere czy WebLogic. Z roku na rok popularność serwera aplikacji rosła, a w 2004 roku nastąpiło przekształcenie spółki JBoss Group LCC, w firmę JBoss Inc. Równolegle do tego wydarzenia nastąpiło wydanie kolejnej, czwartej wersji produktu. Jboss Inc znacznie się rozwinął, a jego wsparcie zostało powiększone o komponenty, które służyć miały serwerom aplikacji. Należały o nich Jboss Rules, jBMP, Jboss Cache czy Hibernate. Po tym, jak firma i produkt zyskały jeszcze większe uznanie i rozpoznawalność, w 2006 roku JBoss Inc wykupiła firma Red Hat Inc. Przy okazji tego wydarzenia wypuszczono nową wersję serwera, czyli Jboss AS 5. Kolejne wersje wypuszczano w następującej kolejności: JBoss AS 6.0 w 2010, JBoss AS 7w 2011, JBoss AS 7.1w 2012 oraz WildFly w 2014 roku.

5.2.3 Struktura WildFly

W architekturze WildFly zastosowana jest konstrukcja modułowa, przy czym należy wyszczególnić ogólne cechy tej technologii:

- Podział na mikrojądro i moduły (wykorzystuje JMX API);
- Można kontrolować wewnętrzne aspekty pracy serwera, programowo lub przez interfejs webowy (JMX Console);
- Istnieje też możliwość dodawania do serwera własnych rozszerzeń (usług) przy użyciu mechanizmu Mbeans [11].

Aby znacznie przyspieszyć proces uruchamiania, zaimplementowano wysoce zoptymalizowany proces rozruchowy, usługi uruchamiane są w tym samym czasie, eliminując niepotrzebne czekanie. Wzięto również pod uwagę lepsze wykorzystanie możliwości procesorów wielordzeniowych. Usługi niekrytyczne utrzymywane zostają w stanie „zamrożenia" do czasu pierwszego użycia. Wykorzystując pamięć podręczną i indeksowane metadane sekwencyjny start znacznie zaoszczędza czas. Wynikiem zastosowanych, powyższych rozwiązań jest dużo szybsze uruchomienie w stosunku do wersji poprzednich – aż 10-krotnie. [12].

WildFly charakteryzuje się radykalnym podejściem do kwestii zarządzania pamięcią. Podstawowe usługi związane z przetwarzaniem zostały skonstruowane w taki sposób, aby zminimalizować alokację stosu. Usługi te opierają się na wspólnych buforowanych indeksowanych metadanych przed duplikowaniem pełnej analizy składniowej, pozwalających na redukcję sterty i migrację obiektów.

Zastosowanie segmentowego ładowania klas zapobiega ich powielaniu, jak również ładowaniu w większym zakresie, niż wymaga tego konfiguracja. Pozwala to nie tylko na zredukowanie górnej granicy pamięci podstawowej, ale również na minimalizację przerw kolektora śmieci (*garbage collectora*). Ponadto konsola administrowania jest w stu procentach bezstanowa i zależna od woli klienta. Startuje natychmiast i nie wymaga jakiejkolwiek pamięci na serwerze.

Połączenie tych optymalizacji pozwala WildFly na działanie z ustawieniami zasobów JVM, również na małych urządzeniach. Pozostawia także większą przestrzeń dla danych aplikacji i pozwala na większą skalowalność.

WildFly wdraża najnowsze biznesowe standardy Javy. Javę EE7 usprawnia wydajność tworzenia aplikacji poprzez zapewnienie bogactwa biznesowych rozwiązań w postaci łatwych w użyciu frameworków, które pozwalają na eliminację standardowego kodu i redukcję utrudnień o charakterze technicznym. Pozwala to zespołom programistycznym na skoncentrowanie się na istotnych kwestiach i potrzebach biznesowych aplikacji. Frameworki składające się na Java EE sprawdzają się świetnie we wspólnym działaniu. Korzystając z tych standardów, nie musimy się martwić o znalezienie magicznych połączeń różnego rodzaju frameworków w taki sposób, aby działały razem w niezakłócony sposób. Budowanie aplikacji w oparciu o standardy pozwala na elastyczność w przypadku migracji pomiędzy różnymi rozwiązaniami. Aplikacje stworzone na bazie powyższych standardów działające na innym serwerze bez problemu można przenieść do WildFly

Hierarchiczne ładowanie klas często sprawia problemy i jest przyczyną nieudanych wdrożeń. Aplikacja JBoss radzi sobie doskonale z ładowaniem klas. Używa modułów JBoss, aby zapewnić izolację aplikacji, każdy komponent jest umieszczany w pliku JAR (pliki te posiadają różne rozszerzenia, zależnie od typu komponentu). Pozwala na ładowanie tylko tych klas, które są aktualnie potrzebne. Możliwe jest także równoczesne ładowanie kilku klas przy wykorzystaniu potencjału procesorów wielordzeniowych.

WildFly został zaprojektowany z myślą o testowaniu. Jego sekretem jest Arquillian, składnik modelu integracji testów, dzięki któremu testy uruchomiają się błyskawicznie wewnątrz rzeczywistego środowiska wykonawczego. Arquillian umożliwia pisanie testów w prawie każdym przypadku użycia aplikacji. Dzięki szybkości WildFly testy Arquillian uruchomiają się prawie tak szybko, jak testy jednostkowe.

5.3. MySQL

MySQL należy do firmy Oracle i jest przez nią ciągle rozwijany. Jest dostępny zarówno na licencji komercyjnej, jak i licencji GPL (Open Source). Do tego projektu został wykorzystany, ponieważ posiada wiele praktycznych rozwiązań i jest obecnie jedną z najszybszych i najbardziej stabilnych baz danych.

5.3.1 Wstęp

MySQL jest serwerem wielodostępnym i wielowątkowym oraz wykorzystuje język zapytań SQL. Ponadto jest solidnym oraz wydajnościowo szybkim systemem do zarządzania relacyjnymi bazami danych. Najważniejszymi zaletami MySQL są: elastyczność, prostota obsługi, szybkość działania, duża wydajność, dostępność kodu źródłowego oraz łatwość konfiguracji.

5.3.2 Historia

System MySQL powstał w połowie lat dziewięćdziesiątych jako następca mSQL. Pierwsza jego wersja została opublikowana w styczniu 1998 r., choć jej korzenie sięgają do roku 1979, do programu UNIREG (którego autorem jest Michael „Monty" Widenius), stworzonego dla szwedzkiej firmy TcX DataKonsult AB. Dziś firmą odpowiedzialną za rozwój jest firma MySQL AB. Programiści MySQL w wrześniu 2001 r. udostępnili wersję alpha 4.0 i kolejno 4.1, która była rekomendowana aż do opublikowania aktualnej wersji 5.0.

5.3.3 Relacyjny model danych

Twórcą teorii relacyjnych baz danych jest Edgar Codd, którego postulaty zostały opublikowane po raz pierwszy w 1970 r. Relacyjna baza danych oznacza bazę powstałą z relacji. Tabela jest podstawowym obiektem bazy danych czyli reprezentacja relacji — technicznego pojęcia matematyki. Oba terminy nie są jednoznaczne, a jedna relacja może być odwzorowana za pomocą wielu różnych tabel. W modelu relacyjnym głównym elementem jest tabela posiadająca daną liczbę kolumn, unikatową nazwę, zawartość oraz nagłówek. Każdy wiersz jest osobnym rekordem informacji. Wszystkie kolumny mają swoją unikatową oddzielną nazwę oraz typ zawartych w nich danych, dzięki któremu wiemy, jakie wartości możemy w niej umieszczać.

W relacyjnym modelu danych tabele muszą posiadać klucz główny, który ma przypisaną unikatową wartość służącą do identyfikacji danego wiersza, oraz mogą posiadać klucz obcy, który wskazuje wiersze w innych tabelach. Rolą kluczy jest zapewnienie, aby dane rekordy były jedyne w swoim rodzaju.

5.3.4 Składnia SQL

Zapytania SQL są najpospolitszymi i najbardziej niezbędnymi operacjami. Poprzez zapytania SQL można przeglądać bazę danych w celu wyszukania potrzebnych informacji. Są one wykonywane przez klauzule „SELECT" oraz „FROM", które są niezbędne. Zapytania SQL mogą być bardziej precyzyjne przy pomocy użycia kilku klauzul, takich jak „WHERE", „ORDER BY" lub „GROUP BY".

Operatory porównania są odpowiedzialne za sprawdzenie, czy dwa wyrażenia są takie same. Mogą być używane na wszystkich wyrażeniach z wyjątkiem wyrażeń text, ntext, lub danych typu image.

MySQL jako system posiada bazę danych oraz relacyjną bazę danych.

W bazie danych przechowujemy główną konstrukcję zbioru danych i możemy do tego zbioru podłączyć wszystko, począwszy od listy zawierającej imiona i nazwiska, poprzez zbiór towarów w dużym supermarkecie, aż po wielkie firmy, które codziennie przetwarzają i przechowują niezliczoną ilość danych i informacji. Kontynuując przykład dużej firmy, która udostępnia, uzupełnia i operuje na tych danych – musi ona oczywiście gdzieś te dane przechowywać, potrzebuje także systemu, aby nimi zarządzać. Takim systemem, który mógłby zarządzać tymi danymi, jest MySQL serwer, który może istnieć jako dodatek do innej aplikacji lub niezależne narzędzie.

W relacyjnej bazie danych, inaczej niż w bazie danych, dane nie są przechowywane w jednym miejscu, a zamiast tego umieszcza się je w oddzielnych tabelach. Taki zabieg wykonuje się w celu zwiększenia szybkości działania, większej wydajności, uproszczenia konfiguracji i poprawienia elastyczności.

Serwer MySQL jako system działa na konstrukcji typu klient – serwer wykorzystywany do przesyłania danych pomiędzy serwerem baz danych a użytkownikiem lub w systemach osadzonych. Serwer baz danych cechuje się przede wszystkim prostotą w użytkowaniu, dużą szybkością, niezawodnością oraz bezpieczeństwem. Pisząc o architekturze Serwer MySQL, trzeba napisać, że składa się z wielowątkowego serwera SQL, który obsługuje narzędzia administracyjne, interfejsy programowania, różnego rodzaju zaplecza aplikacji oraz bibliotek i programów.

5.4. Hibernate

W tej części zostanie przedstawiony framework do realizacji warstwy dostępu do danych. Jego głównym zadaniem jest zapewnienie translacji danych pomiędzy relacyjną bazą danych a światem obiektowym.

5.4.1 Wstęp

Hibernate jest narzędziem Open Source, jest zatem bezpłatny. Na jego wybór zdecydowano się, ponieważ jest to narzędzie stabilne, dojrzałe i dobrze udokumentowane, a co się z tym wiąże, łatwe do nauki. Zasadnicza cześć kodu liczy aż 76 tysięcy linii i jest codziennie pobierana 3000 razy.

5.4.2 Historia

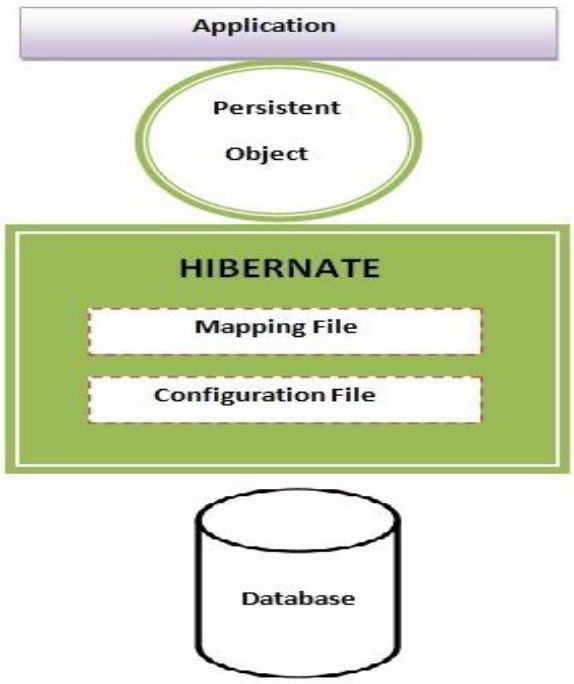
Pierwsze wydanie Hibernate zostało udostępnione w 2001 roku, jego autorami byli Gavin King wraz z kolegami z Cirrus Technologies. Jako cel projektu postawiono sobie uzyskanie lepszych możliwości utrwalania danych niż te oferowane przez EJB2, uproszczenie złożoności oraz uzupełnienie brakujących funkcji.

Na początku roku 2003 zespół Hibernate rozpoczął prace nad wersją Hibernate 2.0 Wersja ta oferowała znacznie więcej ulepszeń w stosunku do poprzedniej. W 2005 roku została wydana wersja 3.0, zaś w grudniu roku 2014 – Hibernate w wersji 4.0. W marcu 2015 roku został wypuszczony update do wersji 5.1.1 final.

Obecnie Hibernate tworzony jest przez firmę Red Hat Jboss. Można go pobrać ze strony http://www.hibernate.org. Znajduje się tam biblioteka Hibernata oraz wykorzystywane przez nią biblioteki innych dostawców. Jest to projekt rozwijany na licencji otwartej.

5.4.3 Architektura Hibernate

Hibernate stanowi warstwę pośredniczącą w komunikacji aplikacji z bazą danych, obsługując trwałość obiektów aplikacji. Ogólną architekturę przedstawia Rysunek 58.



Rysunek 58. Ogólna architektura Hibernate. *Źródło:* [15].

Parametry połączenia z bazą danych i inne opcje, znajdują się w pliku konfiguracyjnym Hibernate, który domyślnie ma nazwę hibernate.properties lub, co jest wygodniejsze, postać pliku XML - hibernate.cfg.xml. Trwałość obiektów Hibernate obsługuje poprzez wykorzystanie informacji o odwzorowaniu obiektowo-relacyjnym modelu danych aplikacji które znajdują się w plikach XML.

Listing 1. Plik konfiguracyjny hibernate.properties. ***Źródło: Opracowanie własne.***

```
hibernate.connection.driverClassName=com.mysql.jdbc.Driver
hibernate.connection.url=jdbc:mysql://localhost:3306/szw
hibernate.connection.username=****
hibernate.connection.password=****
hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.hbm2ddl.auto=create-drop
hibernate.show_sql=true
```

Plik hibernate.properties należy umieścić w ścieżce wyszukiwania klas aplikacji, wtedy zostanie automatycznie wykryty i wczytany w momencie tworzenia obiektu Configuration przez Hibernate. Przedstawiony fragment listingu określa następujące informacje wiersz po wierszu:

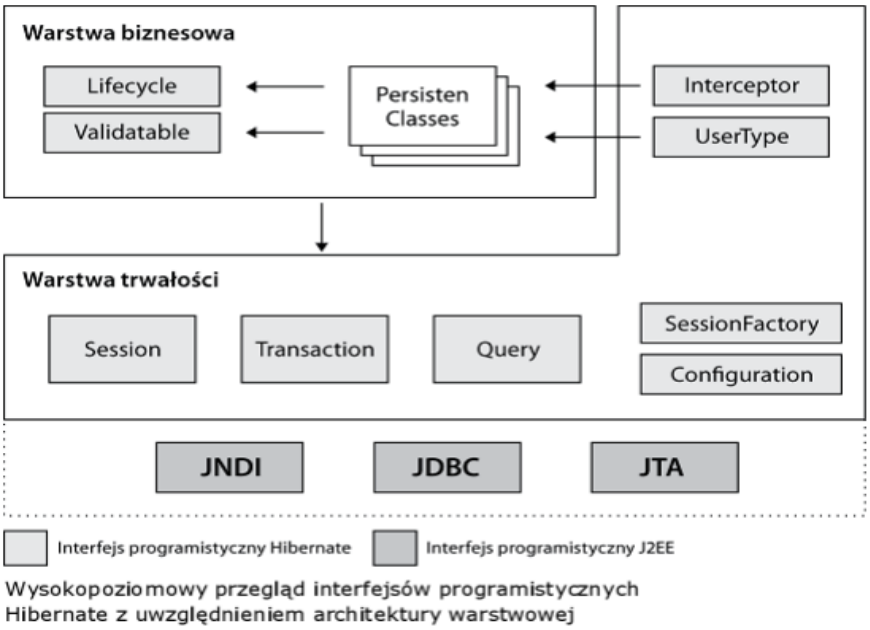
- Nazwę klasy Javy implementującej sterownik JDBC dla konkretnej bazy danych
- Adres URL w formacie JDBC określający adres serwera i nazwę bazy danych dla połączeń JDBC
- Nazwa użytkownika do bazy danych
- Hasło użytkownika do bazy danych
- Klasa dialektu języka SQL używana w bazie danych
- Sposób działania na bazę danych (create, create – drop, update, validate)
- Podgląd tworzonych zapytań

5.4.4 Interfejs Hibernate

Głównym celem projektowym interfejsów była minimalizacji współbieżności pomiędzy poszczególnymi komponentami oprogramowania. Rysunek 59 przedstawia najważniejsze zadania interfejsów w warstwie biznesowej i trwałości.

Hibernate wykorzystuje interfejsy programistyczne Javy w którego skład wchodzą JDBC, który zapewnia odpowiedni poziom abstrakcji funkcji wspólnych dla różnych systemów relacyjnych baz danych, JTA (Java Transaction API) oraz JNDI (Java Naming and Directory Interface). Hibernate dzięki tym interfejsom potrafi skorzystać z dowolnej bazy danych oraz umożliwia integrację z serwerami aplikacji J2EE. Każda aplikacja wykorzystująca Hibernate korzysta z co najmniej pięciu podstawowych interfejsów (Rysunek 59), dzięki którym można pobierać i zapamiętywać trwałe obiekty oraz sterować transakcjami.

- **Interfejs Session** – główny interfejs każdej aplikacji opartej na Hibernate. Koszt utworzenia i zniszczenia sesji jest niewielki. Sesje należy traktować jako bufor lub kolekcje załadowanych obiektów powiązanych z jedną jednostką zdaniową. Nazywany jest także zarządcą trwałości, ponieważ umożliwia dostęp do podstawowych operacji trwałości, takich jak zapis i pobieranie obiektów
- **Interfejs SessionFactory** – w całej aplikacji występuje tylko jeden obiekt SessionFactory, chyba że aplikacja korzysta z wielu baz danych. W takim przypadku potrzeba osobnego obiektu SessionFactory dla każdej z baz danych. Umożliwia wielowątkowa pracę oraz pozwala na tworzenie obiektów klasy Session.



Rysunek 59. Interfejsy Hibernata. *Źródło [14].*

- **Interfejs Configuration** – służy do konfiguracji i uruchomienia Hibernate. Aplikacja używa Configuration do wskazania położenia dokumentów odwzorowań i specyficznych własności dla Hibernate.
- **Interfejs Transaction** – jest interfejsem opcjonalnym. Aplikacje Hibernete nie muszą z niego korzystać, jeżeli zarządzają transakcjami we własnym zakresie. Przeważnie jest używany, aby ukryć szczegóły implementacji konkretnych mechanizmów transakcyjnych.
- **Interfejs Query** – umożliwia wysyłanie zapytań do bazy danych i sterowanie procesami ich wykonania. Tworzenie zapytań odbywa się przy użyciu języka HQL (Hibernate Query Language).

Hibernate Query Language jest pochodną języka SQL. Zapytania są automatycznie przekładane na język SQL. Zorientowany jest obiektowo, a jego zapytania odwołują się do klas, nie tabel. Dzięki powyższym interfejsom oraz HQL możliwe jest zestawienie połączenia oraz wykonywanie operacji na bazie danych.

5.4.5 Podsumowanie

Hibernate to framework zdecydowanie godny polecenia, o czym świadczy liczba jego pobrań, materiałów i pomocy w internecie. Posiada wsparcie dla stylu programowania odpowiedniego dla Javy (Relational Persistence For Idiomatic Java). Cechuje się wysoką wydajnością i skalowalnością , posiada własne API oraz umożliwia stworzenie dużej bazy danych bez większego wkładu pracy. Więcej informacji na temat Hibernate można znaleźć [14] oraz [15].

5.5. Spring

Jest jednym z najpopularniejszych frameworków umożliwiających stworzenie wysoko wydajnościowych, testowalnych aplikacji. Spring wyręcza programistę, dzięki czemu nie ma konieczności kodowania wszystkiego, począwszy od najdrobniejszych użyteczności. Pozwala to na skoncentrowanie się na głównych założeniach projektu [16]. Jedną z unikalnych zalet Springa jest to, że pozwala on na połączenie ze sobą zróżnicowanych komponentów, dzięki czemu jest możliwe tworzenie rozwiązań dla nawet bardzo złożonych problemów.

Należałoby również wspomnieć o innych funkcjonalnościach, takich jak obsługa chmury i wsparcie tradycyjnych systemów zarządzania relacyjnymi bazami danych (RDBMS), jak również NoSQL, lepsza ochrona i kompatybilność z rozwiązaniami mobilnymi. Jak głosi hasło na głównej stronie internetowej, Spring pomaga zespołom programistycznym w tworzeniu prostych, przenośnych, szybkich i elastycznych systemów i aplikacji opartych na JVM (Java Virtual Machine). Pomimo że Spring nie narzuca żadnego konkretnego modelu programistycznego, stał się popularną w społeczności korzystającej z technologii Java alternatywą dla Enterprise JavaBean (EJB).

Framework Spring składa się z wielu modułów zapewniających szereg funkcjonalności. Należą do nich:

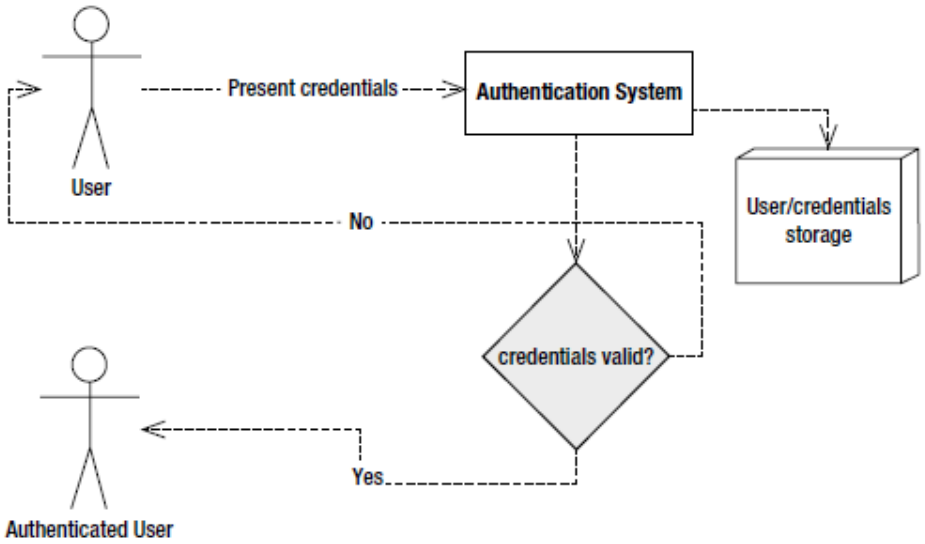
- Kontener rdzenia Spring
- Kontener odwrócenia sterowania (IoC – Inversion of Control)
- Programowanie aspektowe
- Dostęp do danych
- Zarządzanie transakcjami
- Model-Widok-Kontroler (MVC)
- Zdalny dostęp
- Autoryzacja i uwierzytelnianie
- Zdalne zarządzanie (JMX)
- Komunikaty (JMS)
- Obsługa testowania

5.5.1 Spring Security 3.2

Spring Security to wysoce konfigurowalny Framework służący uwierzytelnianiu i kontroli dostępu. Jest w zasadzie standardem w aplikacjach opartych na Spring [18]. Zapewnia zarówno uwierzytelnianie, jak i autoryzację w aplikacjach Java. Cechuje się również ochroną przeciw atakom, zapewnia integrację servletu API oraz umożliwia integrację ze Spring Web MVC.

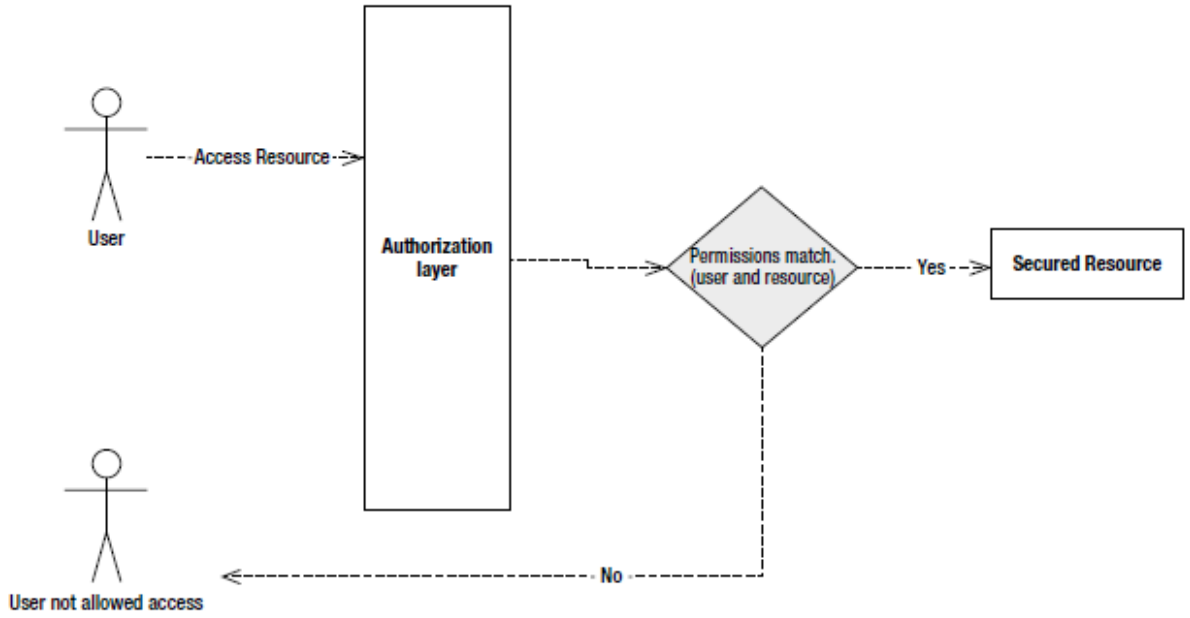
Historia Spring Security rozpoczęła się w 2003 r., kiedy to wystartował projekt „Acegi Security [19]", prowadzony przez Bena Alexa. Projekt został udostępniony publicznie w marcu 2004 r. na licencji Apache, a następnie włączony w portfolio Spring jako oficjalny podprojekt – Spring Security.

Do niniejszej pracy inżynierskiej został wykorzystany z uwagi na swoją kompatybilność z Java, jak również ze względu na poziom bezpieczeństwa. Zapewniają to mechanizmy uwierzytelniania i autoryzacji. Uwierzytelnianie związane jest z upewnieniem się, że użytkownik jest tym, za kogo się podaje. Na poziomie uwierzytelniania Spring zapewnia różne modele uwierzytelniania, takie jak Http Basic authentication, Form Based authentication. Diagram przedstawiający procesy uwierzytelniania przedstawia Rysunek 60.



Rysunek 60. Proces uwierzytelniania. *Źródło: Ibidem.*

Autoryzacja z kolei pozwala na zapewnienie, że użytkownik będzie miał dostęp tylko do tych zasobów, do których jest uprawniony. Proces autoryzacji przedstawia Rysunek 61 Więcej informacji na temat znajdziemy w [20].

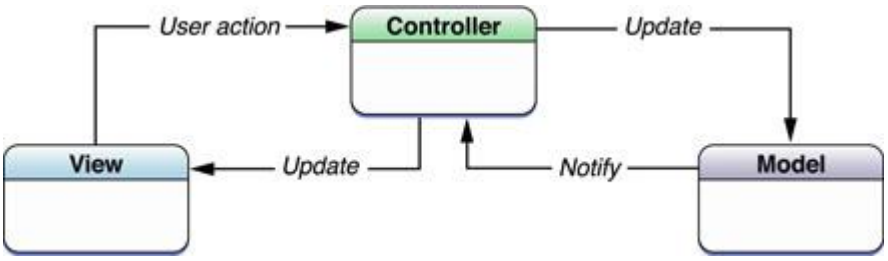


Rysunek 61. Proces autoryzacji. *Źródło: Ibidem.*

5.5.2 Spring MVC

Zanim przejdziemy do opisu Spring MVC, przyjrzyjmy się samemu MVC, czyli Model-View-Controller (z ang. Model-Widok-Kontroler). Pozwoli nam to zrozumieć jego zastosowanie w przypadku omawianego przez nas Frameworku Spring. Czymże zatem jest ów MVC? To wzorzec architektoniczny mający za cel implementację interfejsów użytkowników. Przypisuje on obiektom w aplikacji jedną z trzech ról: modelu, widoku lub kontrolera [21]. Co ciekawe, wzorzec ten nie tylko

przypisuje obiektom role, jakie odgrywać mają w aplikacji, ale również definiuje sposób, w jaki obiekty te mają komunikować się między sobą. Każdy z wymienionych powyżej typów obiektów jest oddzielony od pozostałych przez abstrakcyjne granice, przez które komunikuje się z obiektami pozostałych typów. Schemat funkcjonowania wzorca MVC opisuje poniższy Rysunek 62.



Rysunek 62. Diagram MVC. ***Źródło: [21].***

Tak też, mając na uwadze powyższy rysunek oraz ogólną wstępną informację na temat MVC, możemy pozwolić sobie na przyjrzenie się poszczególnym elementom tego wzorca, w zależności od typu.

Kontroler – obiekty kontrolera są swoistymi pośrednikami pomiędzy obiektami widoku a obiektami modelu aplikacji. Stanowią tym samym niejako łącze, dzięki któremu obiekty widoku dowiadują się o zmianach, jakie zaszły w obiektach modelu oraz odwrotnie. Mogą również wykonywać zadania związane z ustawieniami i koordynacją aplikacji oraz zarządzaniem cyklem życia innych obiektów. Obiekt kontrolera interpretuje działania użytkownika dokonane na obiektach widoku oraz informuje warstwę modelu o nowych lub zmienionych danych. Gdy zmianie ulegają obiekty modelu, obiekt kontrolera informuje o niej obiekty widoku, aby mogły ją wyświetlić użytkownikowi.

Model – obiekty modelu hermetyzują dane charakterystyczne dla aplikacji, definiując jednocześnie logikę i obliczenia pozwalające na manipulację i przetwarzanie danych. Obrazowym przykładem, jaki można wskazać dla obiektów modelu, są obiekty reprezentujące postać w grze czy kontakt w książce adresowej. Obiekt należący do modelu może pozostawać w relacjach jeden do jednego bądź jeden do wielu z obiektami innych modeli. Spora część danych, które stanowią element niezmiennego stanu aplikacji (niezależnie od tego, czy jest on przechowywany w plikach, czy w bazie danych), powinna przebywać w obiektach modelu, gdy dane zostaną załadowane do aplikacji. Ponieważ obiekty modelu reprezentują wiedzę i doświadczenie powiązane z zakresem konkretnego problemu, mogą zostać użyte w innych, podobnych problemach. W idealnej wersji obiekt modelu nie powinien mieć bezpośredniego połączenia z obiektami widoku, które prezentują jego dane i pozwalają użytkownikowi na ich edycję, a więc oddzielone od interfejsu użytkownika i kwestii związanych z prezentacją. Jeżeli chodzi o komunikację, działania użytkownika w warstwie widoku, tworzące bądź modyfikujące dane, są przekazywane przez obiekt kontrolera, skutkując powstaniem bądź aktualizacją obiektu modelu. W przypadku zmiany obiektu modelu powiadamia on obiekt kontrolera, który aktualizuje odpowiednie obiekty widoku.

Widok – obiekty widoku to takie obiekty, które są widzialne dla użytkownika. Obiekt widoku wie, jak powinien wyglądać i reagować na działania użytkownika. Głównym celem obiektów widoku jest reprezentowanie danych z obiektów modelu i umożliwienie ich modyfikacji. Pomimo to obiekty widoku są zazwyczaj oddzielone od obiektów modelu, zgodnie z zasadami wzorca MVC. W związku z wielokrotnym użyciem i konfigurowaniem obiektów widoku zapewniają one spójność pomiędzy aplikacjami. Obiekty widoku są informowane o zmianach w obiektach modelu poprzez obiekty kontrolera w aplikacji i komunikują zmiany zainicjowania przez użytkownika, na przykład tekst wprowadzony do pola tekstowego, również poprzez obiekty kontrolera do obiektów modelu.

Mając wiedzę, czym jest MVC, możemy przejść do zagadnienia Spring MVC. Framework Spring MVC jest zbudowany w oparciu o tzw. DispatcherServlet, który przesyła żądania do procedur obsługowych [17]. Podstawowy element obsługowy jest oparty na adnotacjach *@Controller* oraz *@RequestMapping*, dzięki czemu oferuje szeroki zakres elastycznych metod obsługowych. Sieciowy

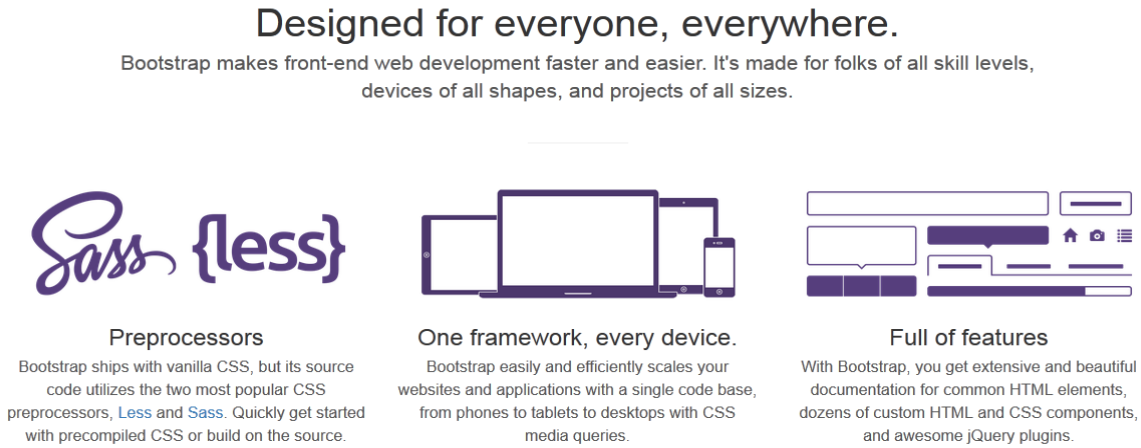
MVC Springa (Spring Web MVC) – *Spring Web Flow* (SWF) stanowi bardzo dobre rozwiązanie zarządzania przepływem stron sieciowych aplikacji.

Należy przy tym choć ogólnie zaznaczyć najciekawsze elementy charakteryzujące ten sieciowy moduł Springa:

- Oddzielenie poszczególnych ról,
- Dająca wiele możliwości a jednocześnie prosta konfiguracja klas Frameworka i aplikacji jako *JavaBeans*,
- Możliwości adaptacyjne, brak inwazyjności i elastyczność,
- Dający możliwość ponownego użycia kod o charakterze biznesowym, bez konieczności duplikacji,
- Indywidualizowane wiązania i walidacja,
- Indywidualizowane mapowanie procedur obsługi i rozdzielczości widoku,
- Elastyczny transfer modelu,
- Indywidualizowane ustawienia regionalne, strefy czasowe, i rozdzielczość tematów, wsparcie dla JSPs, JSTL czy Velocity bez konieczności zapewnienia dodatkowych bibliotek, tworzenia mostów itp.,
- Prosta, lecz potężna biblioteka *tagów* Springa,
- Biblioteka formularzy Spring, znacznie ułatwiająca pisanie formularzy,
- Cykl życiowy ziarna powiązany z żądaniem HTTP w sesji HTTP.

5.6. Bootstrap

W tym miejscu zostanie zaprezentowany framework CSS, który jest odpowiedzialny za dostarczanie czcionek, zbiorów szablonów formularzy, przycisków oraz innych elementów wyświetlanych później na stronach WWW. Bootstrap jest połączeniem HTML, CSS oraz kodu Javascript code zaprojektowanym, aby utworzyć komponenty dla interfejsu użytkownika.



Rysunek 63. Strona główna Boostrap. *Źródło:* [22].

5.6.1 Wstęp

Bootstrap jest darmowym oraz opensource'owym zbiorem narzędzi do tworzenia stron oraz aplikacji webowych. Został zaprogramowany, aby wspierać zarówno HTML5, jak i CSS3. Jego zalety to między innymi: oszczędność czasu, przystosowalność, spójność platformowa, częste aktualizacje.

5.6.2 Historia

Bootstrap, oryginalnie nazwany „Twitter Blueprint", został utworzony w 2011 roku przez zespół inżynierów, w którego skład wchodził Mark Otto oraz Jacob Thornton Twitter. Zasadniczym zamysłem było stworzenie wewnętrznego rozwiązania do rozwiązywania problemów z niespójnościami podczas programowania. Następnie zespół inżynierów z Twittera postanowił zaprojektować platformę bez użycia ustalonej struktury kodu [23].

5.6.3 Architektura Bootstrap

Jak to pokazuje Rysunek 64, do witryny należy dodać 2 pliki. Jeden z plików jest arkuszem CSS, a drugi plikiem JavaScript. Dostępne są ich dwie wersje - zwykła oraz skompresowana, która nie zawiera żadnych zbędnych spacji, znaków nowej linii etc.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

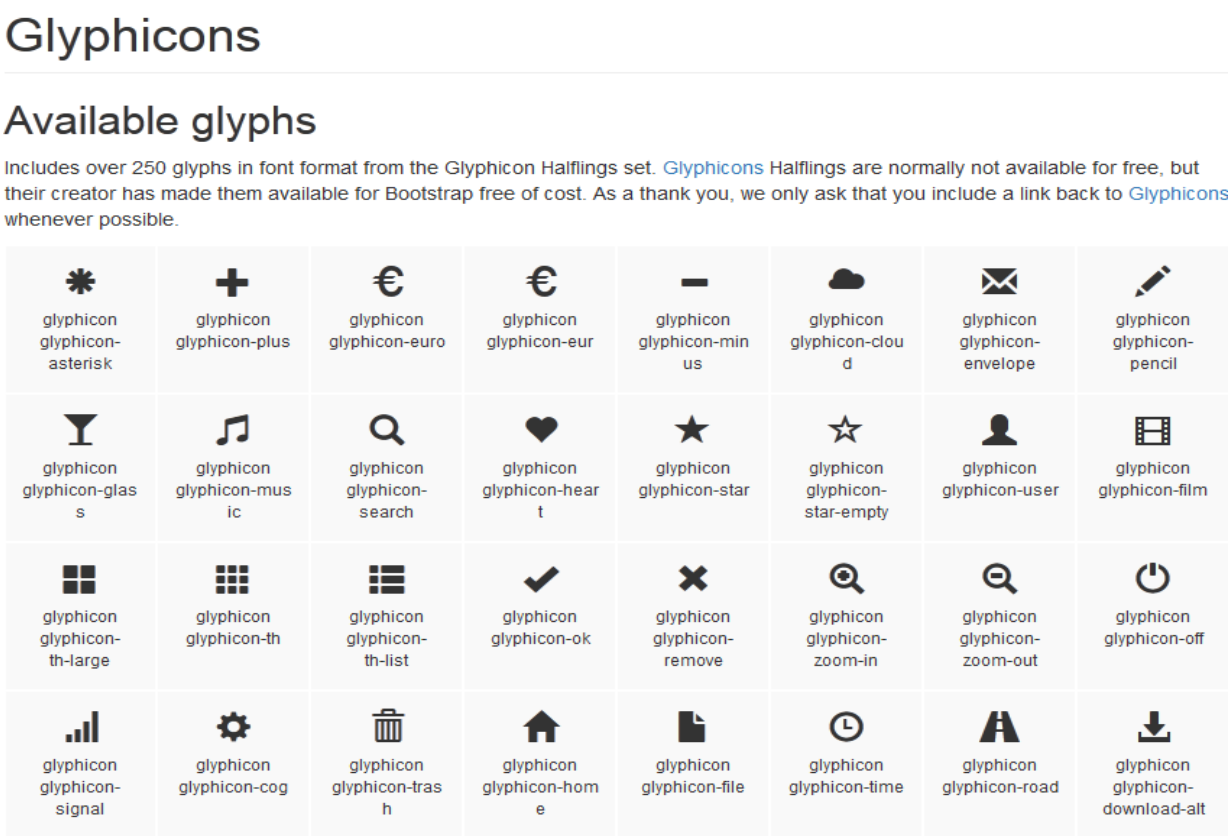
Rysunek 64. Struktura plików Bootstrap. ***Źródło:*** **[22].**

5.6.4 Zawartość framework

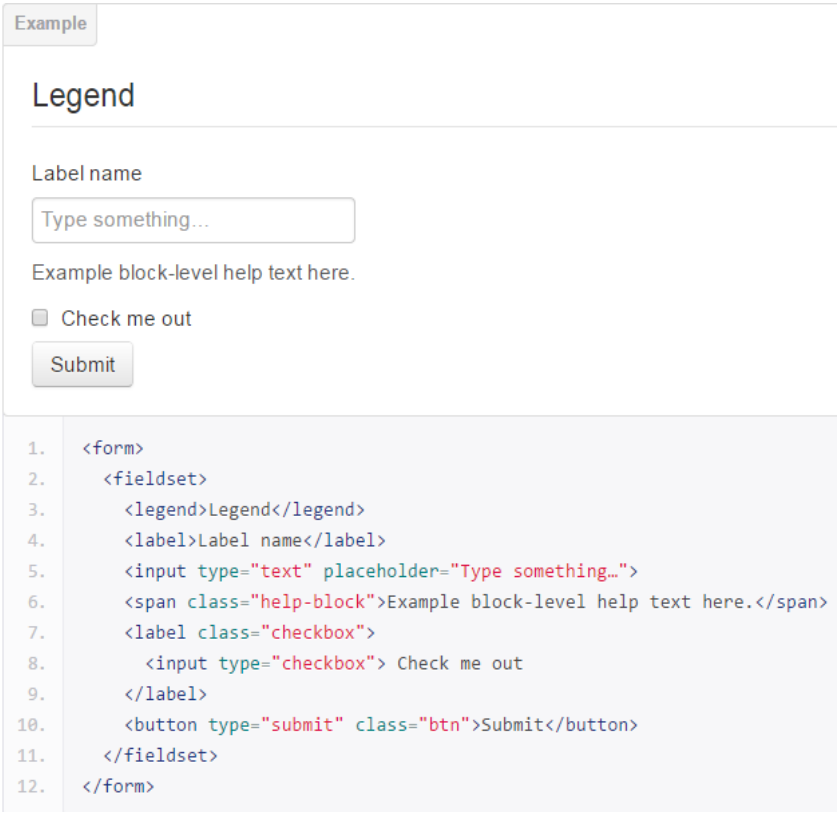
Twitter Bootstrap posiada wszystkie potrzebne elementy, które może wykorzystać twórca stron internetowych – począwszy od budowy w HTML5 po formatowanie w zintegrowanym języku CSS3, na dodatkowych komponentach wykorzystujących JavaScript (jQuery). Autorzy narzędzia podzielili jego wyposażenie na cztery główne kategorie:

- **Scaffolding** – Dosłownie „rusztowanie", na którym opiera się framework. Zdefiniowane są tutaj globalne style dla strony, zresetowane domyślne ustawienia CSS'owe przeglądarek oraz kolor tła i odnośników. Dodatkowo znajdziemy tu ustawienia dotyczące systemu Grid Layout (Grid 63).
- **Base CSS** – Tutaj znajdziemy opracowane style dla wszelkich elementów z języka HTML5, takich jak formularze (Rysunek 66), tabele czy buttony. Dodatkowo zdefiniowano też klasy, zawierające bardzo ciekawe ikony z paczki Glyphicons (Rysunek 65).

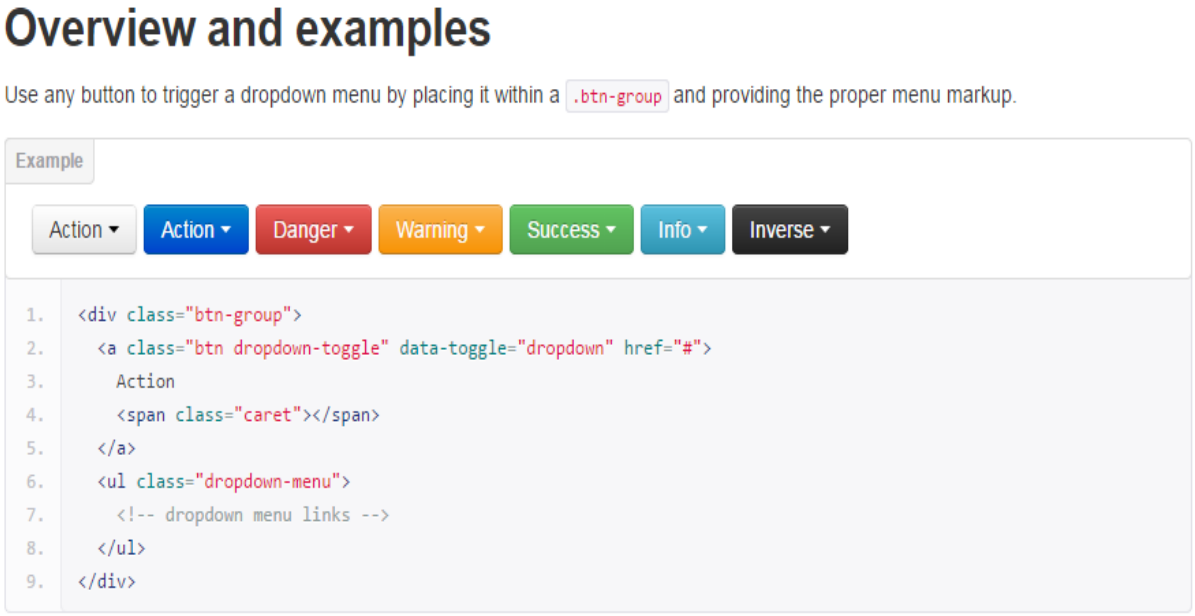
- **Components** – W tej części zgromadzono podstawowe style dla elementów interfejsu, takich jak karty z treścią, panele nawigacyjne, okienka alertów (informacyjne, ostrzegawcze, błędu), nagłówki i wiele, wiele innych. Rysunek 67 przedstawia przykładową grupę przycisków, w tym przypadku dropdown menu.
- **Javascript plugins** – Jeżeli potrzebujemy interaktywnych elementów, takich jak slidery, okienka akcji, tooltipy, rozwijane menu, to właśnie te elementy zdefiniowano w tej kategorii.



Rysunek 65. Ikony Glyphicons. *Źródło:* [24].



Rysunek 66. Formularz z kodem. *Źródło:* [24].



Rysunek 67. Button dropdown menu. *Źródło:* [25].

5.6.5 Grid

CSS Grid, znajdujący się w Bootstrap, zawiera 12-kolumnowy (stały lub płynny) zestaw styli służący do poprawnego wyświetlania elementów na różnorodnych wielkościach ekranu (*Responsive*

design). Dostosowuje się do szerokości pomiędzy 724 a 1170 pikseli, w zależności od rozdzielczości ekranu. Rysunek 68 przedstawia rozkład przykładowego Grida.

Rysunek 68. Przykładowy Grid. *Źródło* [22].

5.6.6 Szkielet szablonu

Jak zostało wspomniane wcześniej, wystarczy podpiąć dwa pliki – arkusz CSS i Pluginy JS. W dużym skrócie wygląda to mniej więcej tak, jak to przedstawia Rysunek 69.

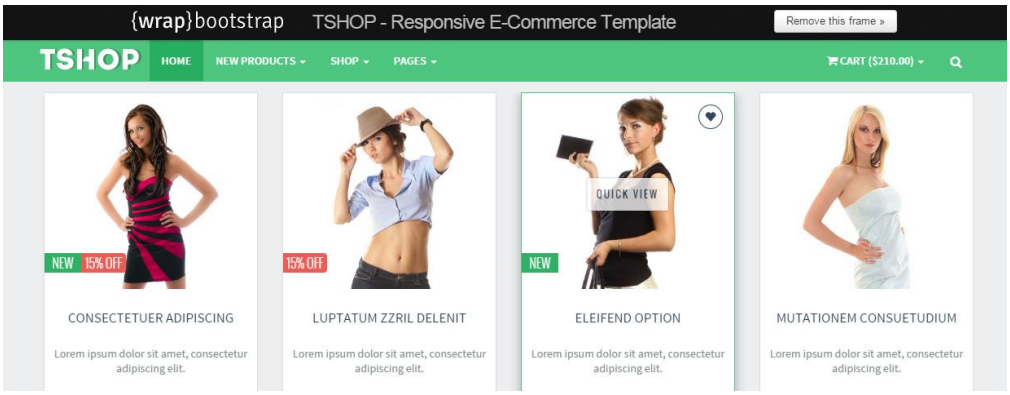
```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="http://code.jquery.com/jquery.js"></script>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Rysunek 69. Bootstrap Template. *Źródło* [22].

5.6.7 Przykłady stron Bootstrap

Poniższe zdjęcia przedstawiają strony internetowe (Template), w których zastosowano technologię Bootstrap.

Rysunek 70. Przykład strony wykorzystującej Bootstrap. *Źródło:* [28].



Rysunek 71. Przykład strony wykorzystującej Bootstrap. *Źródło:* [28].

5.7. Maven 3

Maven, a właściwie Apache Maven, jest wszechstronnym narzędziem służącym do zarządzania projektem programistycznym na licencji Apache.

Został stworzony w 2002 r. w ramach podprojektu Apache Turbine przez Takari's Jason van Zyl. W 2003 r. projekt został zaakceptowany przez Fundację Apache jako oficjalny projekt o wysokim znaczeniu. W 2004 r. udostępniono Maven 1.0, natomiast niewiele ponad rok później Maven 2.0.W roku 2010 na rynek wyszedł Maven 3.0, w pełni kompatybilny z wersją poprzednią.

Bazując na koncepcji modelu obiektu projektu [30] pozwala na zarządzanie budową projektu, raportowaniem oraz dokumentacją. Pozwala na automatyzację budowy projektu, realizując swoje zadania w oparciu o pobierane przy pierwszym użyciu wtyczki. Co ważne, Maven zapewnia dwa aspekty budowy oprogramowania. Po pierwsze, opisuje, jak zbudowana jest aplikacja, po drugie zaś, określa jej zależności. Pomimo współpracy z wieloma językami programowania najlepiej współpracuje z wykorzystaną w niniejszym projekcie Javą, dynamicznie pobierając jej biblioteki.

POM, czyli *Project Object Model*, to dokument XML, który opisuje projekt. Zawiera szczegóły budowy projektu, może także przechowywać informacje o zespole programistów oraz zastosowanych systemach wspomagających rozwój oprogramowania. Programista może plik POM.xml napisać samodzielnie od podstaw bądź użyć Mavena, za którego pomocą można wygenerować go automatycznie. Listing 2 zawiera kod źródłowy pliku POM.xml napisany przez programistę ręcznie.

Listing 2. Kod z pliku POM.xml. *Źródło: Opracowanie własne.*

```
    projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
          <!-- Wersja poma -->
           <modelVersion>4.0.0</modelVersion>
          <!-- kategoria maven -->
           <groupId>pl.pja.edu.io2014</groupId>
          <!-- artefakt -->
           <artifactId>szw</artifactId>
          <!-- projekt tworzy archiwum WAR -->
           <packaging>war</packaging>
          <!-- wersja -->
           <version>0.1-SNAPSHOT</version>
          <!-- ludzka nazwa projektu -->
           <name>System Zarządzania Wspólnotami</name>
          <!-- lokalizacja aplikacji -->
           <url>http://10.4.4.140/szw</url>
```

```
        <!-- stałe projektu -->
    <properties>
            <!-- wersja Javy modułu -->
             <java.version>1.8</java.version>
            <!-- kodowanie plików źródłowych -->
             <project.build.sourceEncoding>UTF
        8</project.build.sourceEncoding>
            <!-- wersja Spring -->
             <spring-framework.version>4.1.2.RELEASE</spring-
        framework.version>
            <!-- wersja Spring security -->
             <spring-security.version>3.2.5.RELEASE</spring-
        security.version>
            <!-- wersja Hibernate -->
             <hibernate.version>4.3.7.Final</hibernate.version>
            <!-- wersja Logback -->
             <logback.version>1.0.13</logback.version>
            <!-- wersja SLF4J -->
             <slf4j.version>1.7.5</slf4j.version>
            <!-- wersja AspectJ -->
             <aspectj.version>1.8.4</aspectj.version>
    </properties>
```

Listing 3 przedstawia plik logowania w czasie generowania aplikacji webowej.

Listing 3. Generowanie aplikacji Maven. ***Źródło: Opracowanie własne.***

```
    mvn clean install -Pdev
    [INFO] Scanning for projects...
    [INFO] ------------------------------------------------------------------
    [INFO] Building System Zarządzania Wspólnotami 0.1-SNAPSHOT
    [INFO] ------------------------------------------------------------------
    [INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ szw ---
    [INFO] Deleting J:\edu\git\szw\target
    [INFO] --maven-resources-plugin:2.5:resources(default-resources) @ szw
    [debug] execute contextualize
    [INFO] Using 'UTF-8' encoding to copy filtered resources.
    [INFO] Copying 1 resource
    [INFO] Copying 4 resources
    [INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ szw
    [INFO] Changes detected - recompiling the module!
    [INFO] Compiling 158 source files to J:\edu\git\szw\target\classes
    [INFO]
    /J:/edu/git/szw/src/main/java/pl/edu/pja/io2014/szw/core/dao/PersonDAO
    .java: Some input files use unchecked or unsafe operations.
    [INFO]
    /J:/edu/git/szw/src/main/java/pl/edu/pja/io2014/szw/core/dao/PersonDAO
    .java: Recompile with -Xlint:unchecked for details.
    [INFO]-maven-resources-plugin:2.5:testResources(default-
    testResources)@szw-
    [debug] execute contextualize
    [INFO] Using 'UTF-8' encoding to copy filtered resources.
    [INFO] Copying 0 resource
    [INFO]-maven-compiler-plugin:3.2:testCompile(default-testCompile)@szw-
    [INFO] Nothing to compile - all classes are up to date
    [INFO] --- maven-surefire-plugin:2.10:test (default-test) @ szw ---
    [INFO] Surefire report directory:J:\edu\git\szw\target\surefirereports
    -------------------------------------------------------
     T E S T S
    -------------------------------------------------------
```

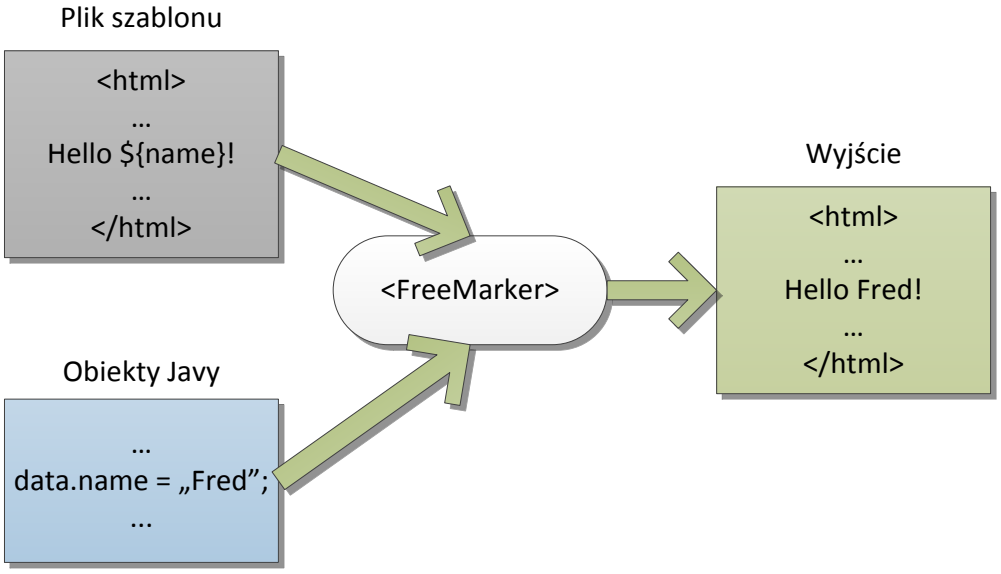
```
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO] --- maven-war-plugin:2.3:war (default-war) @ szw ---
[INFO] Packaging webapp
[INFO] Assembling webapp [szw] in [J:\edu\git\szw\target\szw]
[INFO] Processing war project
[INFO] Copying webapp resources [J:\edu\git\szw\src\main\webapp]
[INFO] Webapp assembled in [2106 msecs]
[INFO] Building war: J:\edu\git\szw\target\szw.war
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ szw
[INFO]Installing J:\edu\git\szw\target\szw.war to
J:\edu\repo\pl\pja\edu\io2014\szw\
0.1-SNAPSHOT\szw-0.1-SNAPSHOT.war
[INFO] Installing J:\edu\git\szw\pom.xml to
J:\edu\repo\pl\pja\edu\io2014\szw\
0.1-SNAPSHOT\szw-0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
[INFO] Total time: 16.886s
[INFO] Finished at: Tue Apr 14 20:26:14 CEST 2015
[INFO] Final Memory: 20M/192M
[INFO] ------------------------------------------------------------------
```

Ponadto Maven udostępnia mechanizmy dziedziczenia i agregacji pomów oraz pozwala na tworzenie własnych artefaktów. Podsumowując, Maven jest narzędziem rozbudowanym i jednocześnie bardzo prostym w podstawowym użyciu, dzięki czemu jest przydatny nawet w małych projektach. Niewątpliwie użycie go w tym projekcie zdecydowanie ułatwiło pracę.

5.8. FreeMarker

FreeMarker jest systemem szablonów, mającym za cel generację na wyjście tekstu opartego na szablonach. To bezpłatne oprogramowanie na licencji Apache. Jest biblioteką klasową dla programistów posługujących się językiem Java. Sam w sobie nie jest aplikacją, która byłaby wykorzystywana przez użytkownika końcowego. Jednakże programista może włączyć go w strukturę swojego produktu. FreeMarker został opracowany w sposób umożliwiający praktyczne zastosowanie dla tworzenia stron internetowych opartych o HTML, głównie dzięki aplikacjom serwletowym, odzwierciedlającym wzór MVC (5.5.2) [31].

Koncepcja użycia wzoru MVC dla dynamicznych stron internetowych wiąże się z oddzieleniem od siebie projektantów czy autorów HTML od programistów. Pozwala to na intensyfikację efektów, poprzez umożliwienie każdemu z twórców pracy w swojej dziedzinie, w której ma największą wiedzę i doświadczenie. Twórcy HTML mają możliwość zmiany wyglądu strony bez konieczności angażowania w to programistów (którzy musieliby ponownie skompilować kod), ponieważ logika aplikacji, czyli programu Java jest oddzielona od szablonów FreeMarker. Taka rozdzielność jest bardzo użyteczna nawet w przypadku, gdy twórca HTML i programista jest tą samą osobą, ponieważ pozwala na utrzymywanie czystości i łatwości obsługi aplikacji. Nietrudno zatem zrozumieć, dlaczego FreeMarker został wykorzystany i okazał się tak dalece użyteczny w przypadku niniejszej pracy inżynierskiej, wykonywanej jako projekt zespołowy. Nie mnie,j jednocześnie podkreślić należy, że choć FreeMarker posiada potencjał programistyczny, nie jest pełnym językiem programistycznym. Zapewnia jedynie generację tekstu odzwierciedlającego dane wytworzone przez programy Javy. Wyjaśnienie zasady działania na Rysunek 72.



Rysunek 72. Jak funkcjonuje FreeMarker. ***Źródło:*** **[31].**

FreeMarker może zostać użyty dla wygenerowania każdego rodzaju tekstu – HTML, XXML, RTF, kod źródłowy Java itd. To, na co należy przy tym jeszcze zwrócić uwagę, to fakt, że FreeMarker nie może w żadnym razie być uznanym za Framework aplikacji sieciowych. Jest wygodnym elementem takich frameworków ale sam w sobie nie ma pojęcia o serwetach czy http. Po prostu zajmuje się generowaniem tekstu, a tym samym jest bardzo dobrym narzędziem również do aplikacji nie będących sieciowymi. Rzecz jasna, w przypadku niniejszej pracy inżynierskiej znalazł on pełne zastosowanie, ułatwiając tworzenie i zarządzanie elementami aplikacji.

5.9. Eclipse

Eclipse to wielojęzykowe zintegrowane środowisko programistyczne oparte na otwartej licencji. Framework służy głównie tworzeniu złożonych aplikacji działających po stronie klienta. Pozwala na tworzenie aplikacji działających pod popularnymi systemami operacyjnymi, takimi jak Windows, Safari czy Linux. Pozwala na korzystanie z szerokiej gamy języków programowania, ogromną zaletą jest też jego rozszerzalność przy użyciu różnego rodzaju wtyczek.

Ponieważ Eclipse znalazł swoje zastosowanie w toku tworzenia niniejszej pracy inżynierskiej, poświęćmy mu nieco więcej uwagi. A zatem czym dokładnie jest Eclipse i skąd się wziął?

Powstał w ramach i na użytek IBM, a właściwie Object Technology International (OTI) funkcjonującego w strukturze tej organizacji [28]. IBM zamierzał zredukować liczbę konkurencyjnych, niekompatybilnych środowisk programistycznych oferowanych klientom, jednocześnie zwiększając stopień ponownego użycia wspólnych komponentów tych środowisk. Kompatybilność i możliwości ponownego użycia miały znacznie ułatwić pracę zespołom programistycznym, które mogłyby wymieniać pomiędzy sobą poszczególne komponenty i łączyć je ze swoimi projektami. Tak zaczął powstawać Eclipse. Oczywiście nie powstał jednego dnia, a stopniowo ewoluował z istniejących już środowisk, takich jak IBM VisualAge for Smalltalk™ i IBM VisualAge for Java™ [29], obydwa stworzone w Smalltalk.

Utworzono niewielką grupę ekspertów, która bazując na doświadczeniach płynących z poprzednich środowisk programistycznych, miała stworzyć nowe, lepsze narzędzie. Tak powstał Eclipse. Jego wielką zaletą była już wówczas rozszerzalność przez instalowanie dodatkowych wtyczek, a otwartość źródła i darmowa licencja umożliwiająca dostęp szerszemu gronu odbiorców były naturalną drogą rozwoju.

Projekt Eclipse bazujący na otwartej licencji został zapowiedziany oficjalnie w 2001 r. przez grupę spółek tworzących tzw. Konsorcjum Eclipse (*Eclipse Consortium*), które przeszło reorganizację do modelu *non-profit* w lutym 2004 r. [30]. W ten sposób mały początkowo projekt rozrósł się w szereg powiązanych ze sobą projektów, które ustanowiły podstawę dziesiątek aplikacji o charakterze komercyjnym. Projekt Eclipse jest ciągle rozwijany, powstają jego kolejne edycje.

Napisany głównie w języku Java Eclipse stanowi zatem zintegrowane środowisko programistyczne (*integrated development environment* – IDE), którego modyfikowaniu służy przestrzeń robocza (tzw. *workspace*) i rozszerzalny system wtyczek. (wtyczki te mogą służyć tworzeniu aplikacji w innych niż Java językach, ale ze względu na tematykę niniejszej pracy nie ma potrzeby rozwijania tego wątku).

Architektura wtyczek pozwala na pisanie wszelkich pożądanych rozszerzeń tego środowiska. Java i *Concurent Versions System* (CVS) obsługiwane są przez Eclipse SDK (*Software Development Kit*), podobnie jak obsługa systemów kontroli wersji w przypadku wtyczek zewnętrznych. SDK zawiera narzędzia programistyczne Java (Eclipse Java development tools – JDT), w których skład wchodzi z kolei wbudowany kompilator (przyrostowy – ang. *incremental*) i pełen model plików źródłowych języka Java. IDE, korzystając z zestawu metadanych z przestrzeni roboczej pozwala na modyfikację zewnętrznych plików, jeśli tylko właściwy zasób z przestrzeni roboczej zostanie odświeżony.

5.10. Redmine

Redmine jest aplikacją internetową służącą do zarządzania projektami. Został napisany z wykorzystaniem technologii Ruby on Rails. Aplikacja jest na licencji otwartej i wydana zgodnie z GNU General Public License w wersji 2. Rysunek 73 przedstawia wygląd strony zaraz po zalogowaniu.

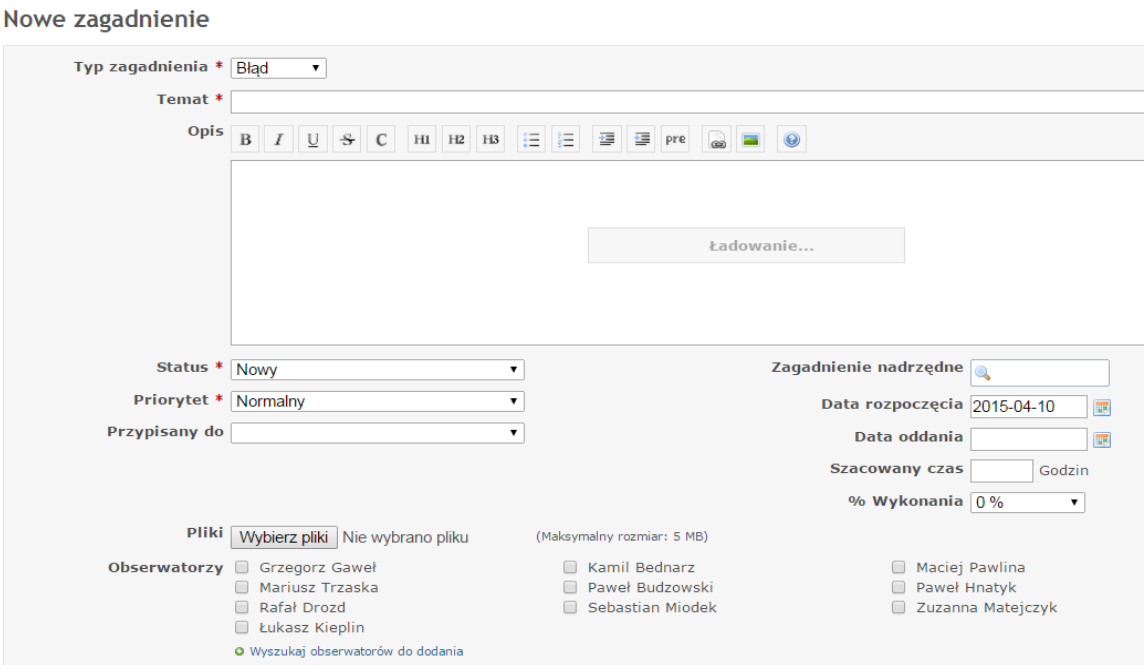
Rysunek 73. Redmine. ***Źródło: Opracowanie własne.***

Redmine jako narzędzie pracy już w minimalnej instalacji dostarcza niezbędnych narzędzi, które pozwalają na zarządzanie zadaniami w projekcie. Dzięki aplikacji, oprócz wspomnianej wyżej funkcjonalności, możemy przydzielać zadania osobie z zespołu, śledzić i rozliczać czas ich wykonywania oraz raportować.

Jednym z najważniejszych dla naszego projektu jest moduł zagadnień. Zagadnienia (Rysunek 74) określają listę rzeczy do zrobienia. Domyślnie rodzaje zagadnień podzielono na trzy typy:

- Zadanie reprezentuje zadanie do wykonania.
- Wsparcie ukazuje cechę rozwiązania, jaką użytkownicy chcieliby zobaczyć.

- Błąd przedstawia błędy napotkane podczas funkcjonowania projektu.

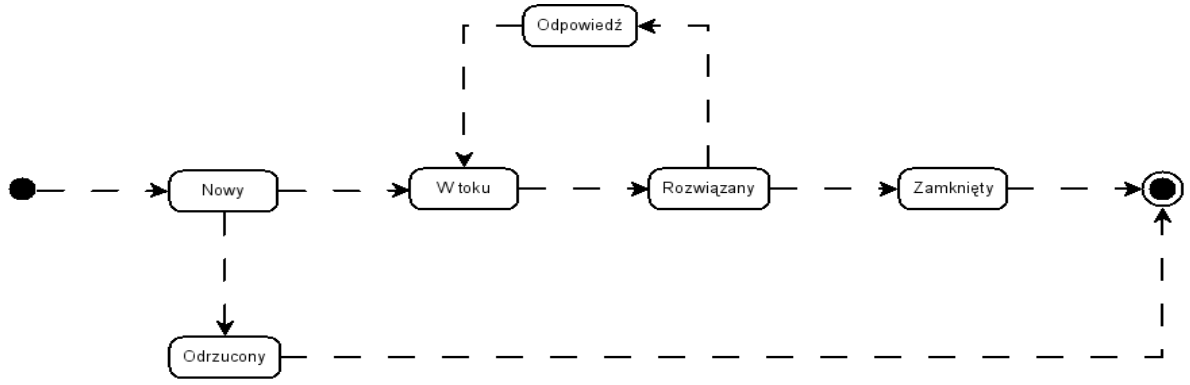


Rysunek 74. Panel nowych zagadnień. ***Źródło: Opracowanie własne.***

Lista zagadnień z określonymi typami pozwala na określenie, na jakim etapie jest projekt i czy się rozwija. Aby ułatwić aktualizowanie zagadnień, wprowadzono sześć statusów:

- Nowy – zagadnienie wprowadzone do systemu, jeszcze nikt nie rozpoczął nad nim prac;
- W toku – osoba, której powierzono zagadnienie, zapoznała się z nim i rozpoczęła nad nim pracę, status utrzymywany jest do czasu zakończenia realizacji i przekazania do testów;
- Rozwiązany – zagadnienie zostało zrealizowane i przekazane do weryfikacji;
- Odpowiedz – pojawienie się zapytania lub komentarza do zagadnienia, wymagana jest reakcja zgłaszającego lub realizatora zagadnienia, odpowiedz sugeruje braki w rozwiązaniu;
- Zamknięty – zagadnienie zostało zrealizowane i zweryfikowane, akceptacja zagadnienia;
- Odrzucony – zagadnienie zostało odrzucone i nie zostanie zrealizowane, może tego dokonać tylko kierownik projektu.

Redmine dopuszcza dowolny system przejść pomiędzy stanami, ale zaleca się stosowanie zasad, które przedstawia Rysunek 75.



Rysunek 75. Statusy zagadnień. ***Źródło*** **[31].**

Redmine posiada duże możliwości konfiguracyjne – począwszy od organizacji projektu poprzez nadawanie uprawnień użytkownikom kończąc na dodatkowych funkcjach, które są dostępne w postaci pluginów na oficjalnej stronie [32].

Do głównych zalet należy zaliczyć:

- Obsługę wielu projektów, stworzenie hierarchicznej struktury projektów i podprojektów oraz archiwizację zakończonych;
- Elastyczną kontrolę dostępu na podstawie roli, użytkownicy mogą mieć przypisany różny poziom uprawnień do różnych projektów;
- System śledzenia zagadnień i problemów;
- Wykres Gantt i kalendarz;
- Możliwość definiowania nowych typów zagadnień i ich statusów;
- Aktualności, dodawanie plików i dokumentów;
- Możliwość zakładania forów dyskusyjnych do poszczególnych projektów;
- Rejestrowanie czasu pracy, a co się z tym wiąże, kontrolę nad kosztami;
- Integrację z systemami kontroli wersji;
- Powiadomienia e-mail;
- Wiele wersji językowych, w tym polska.

Według naszej opinii, opartej na użytkowaniu systemu do stworzenia projektu, jest narzędziem godnym polecenia. Dzięki swojej elastyczności połączonej z szerokim zakresem funkcji pozwala określić aktualny stan, dzięki czemu jest niezwykle pomocny dla kierownika projektu. Więcej informacji znajdziemy na oficjalnej stronie produktu [32].

6. Projekt i implementacja

Celem rozdziału jest przedstawienie etapu projektowania i implementacji systemu zarządzania wspólnotami mieszkaniowymi.

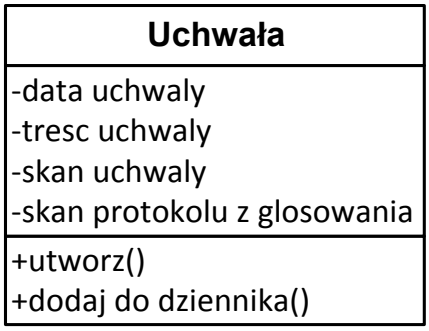
6.1. Wstęp

Głównym zadaniem fazy projektowania jest opracowanie szczegółowego planu implementacji systemu. Projekt musi być wystarczająco szczegółowy, aby mógł być podstawą do implementacji, a stopień jego szczegółowości powinien być dopasowany do poziomu zaawansowania programistów. Kolejną fazą jest faza implementacji, której wynikiem powinna być działająca aplikacja. Niemniej jednak należy pamiętać, że poprawna analiza oraz właściwy projekt są podstawą pozytywnego zakończenia implementacji.

6.2. Diagramy klas

Rysunek na kolejnej stronie przedstawia diagram klas (Rysunek 78) przedstawiający logikę biznesową zaprojektowanej aplikacji. Diagram opisuje strukturę systemu w kategoriach klas i obiektów. Pokazuje kluczowe elementy (klasy) oraz ich związki w systemie. Służy do przedstawienia współpracy klas oraz pozwala na sformalizowanie specyfikacji danych i zestawu metod. Służy jako graficzny środek pokazujący szczegóły implementacji.

W diagramach UML klasy i obiekty reprezentowane są poprzez prostokąty (Rysunek 76) podzielone w pionie na trzy części. Górna zawiera nazwę klasy (obiektu), środkowa przedstawia jego atrybuty, a końcowa cześć prezentuje operacje (metody).

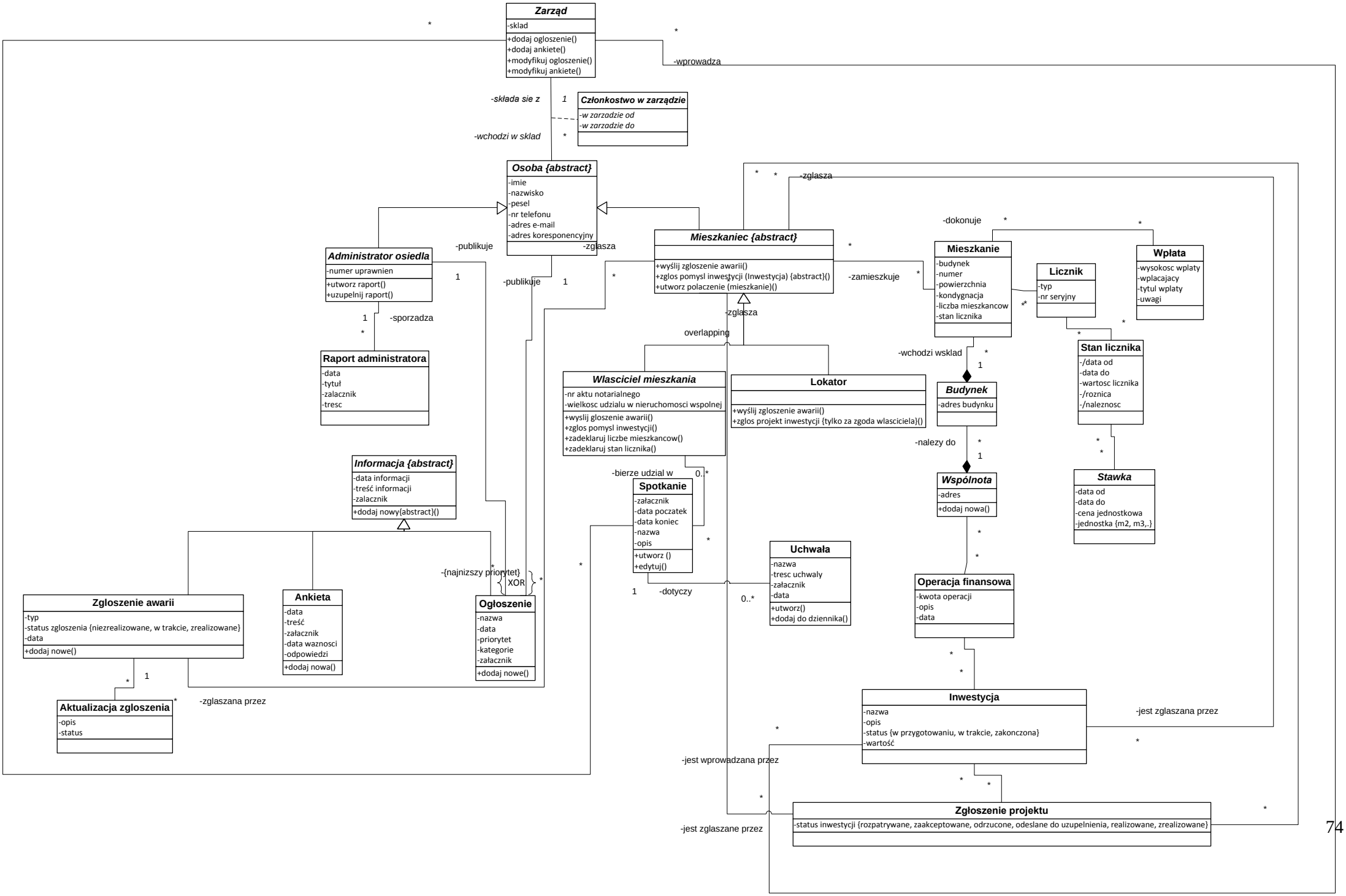


Rysunek 76. Fragment diagramu klas – Uchwała. ***Źródło: Opracowanie własne.***

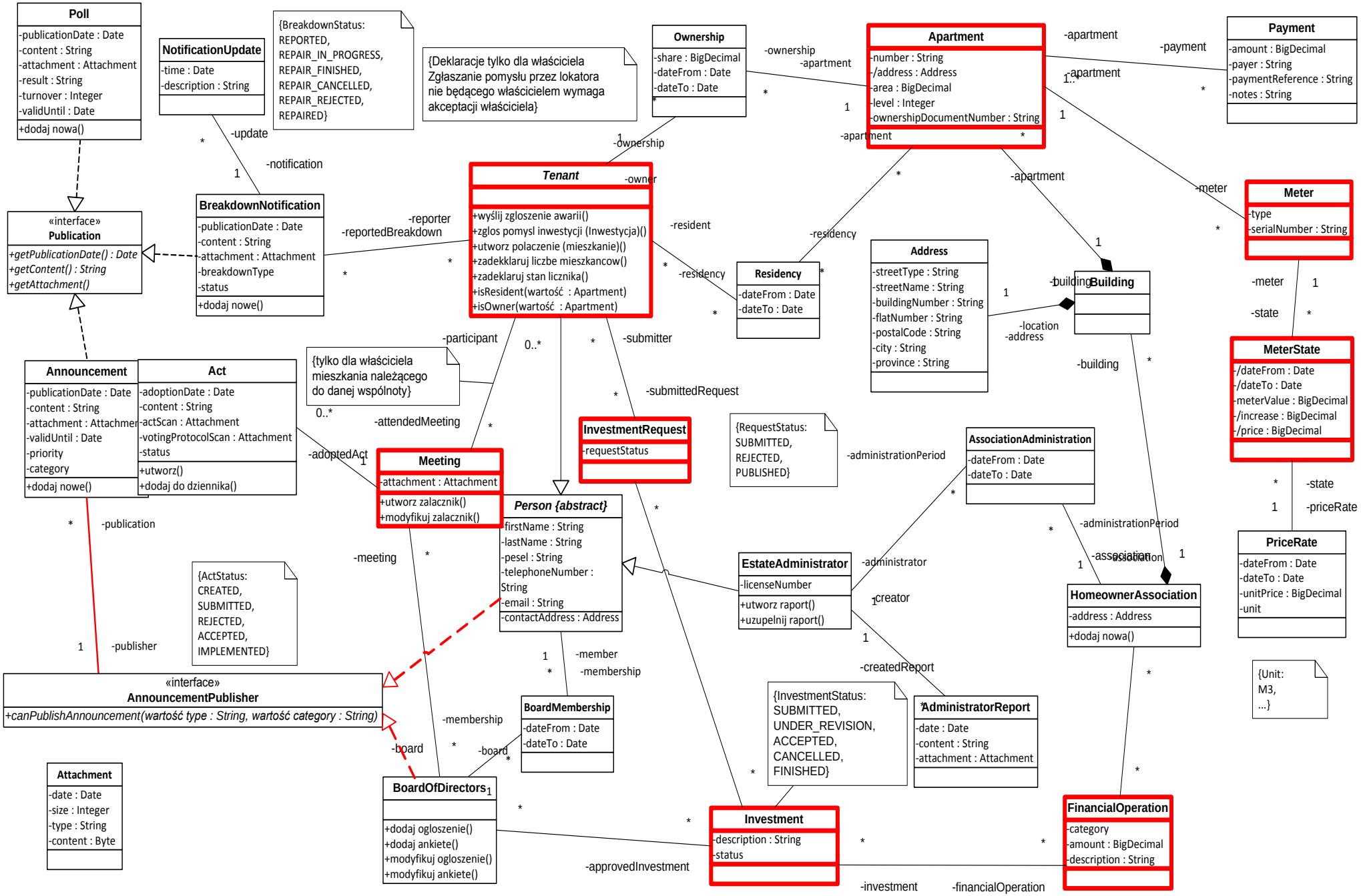
Projektowy diagram klas zawiera Rysunek 79. Różni się od zwykłego diagramu klas tym, że jest uszczegółowiony, wszystkie konstrukcje nieistniejące w języku zostały zastąpione, został również uzupełniony o metody, które wynikły w procesie analizy dynamicznej. Podjęto następujące decyzje projektowe:

- Zgłaszający dla ogłoszenia (xor między Zarząd/BoardOfDirectors, Administrator/EstateAdministrator i Mieszkaniec/Tenant) zostanie zastąpiony instancją implementującą interfejs AnnouncementPoster. Interfejs będzie dołączany do klas Person i BoardOfDirectors i implementowany w klasach BoardOfDirectors, Tenant, EstateAdministrator.
- Klasy Lokator i Właściciel zostały zastąpione przez
    - Asocjacje między klasą Tenant i klasami Apartment w dwóch rolach: ownership i residency

    - Metody weryfikujące, czy osoba jest (była) lokatorem lub właścicielem danego mieszkania: isOwner i isResident
    - Ograniczenia na metody instancyjne klasy Tenant, uzależniające możliwość wykonania danej operacji od powiązania danej osoby z danym mieszkaniem.
- Założono że EstateAdministrator i Tenant mogą się dublować, tzn. może istnieć osoba istniejąca w obu rolach jednocześnie. Nie implementujemy w tym wypadku overlappingu. Zakładamy, że będą dwie instancje klasy Person. Oznacza to w szczególności duplikowanie loginów itd.
- Usunięcie zmiennych stan licznika i liczba mieszkańców z mieszkania – wynikają z asocjacji.
- Klasy Ankieta, Zgłoszenie Awarii i Ogłoszenie zostały spłaszczone do oddzielnych obiektów, przemawia za tym brak wspólnego użycia na poziomie publikacji oraz brak instancyjnych metod przeładowanych (klasowe i tak w Javie nie podlegają dziedziczeniu). Powstały w związku z tym klasy Poll, BreakdownNotification oraz Announcement oraz interfejs Publication.
- Wszystkie ograniczenia w postaci wyliczeń (statusy, jednostki, kategorie) są implementowane przy użyciu enumeracji.



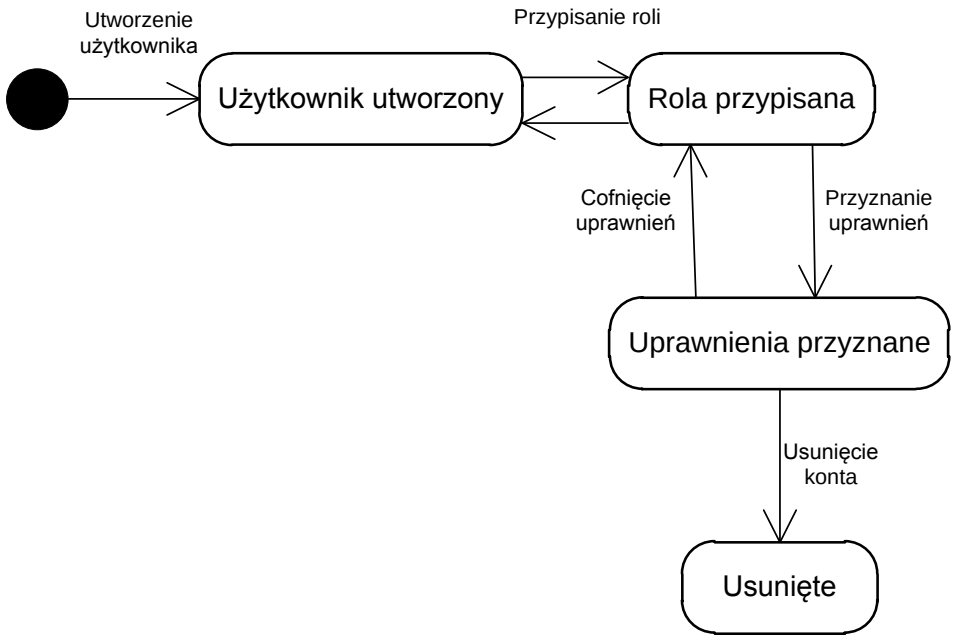
Rysunek 77. Analityczny diagram klas. *Źródło: Opracowanie własne.*



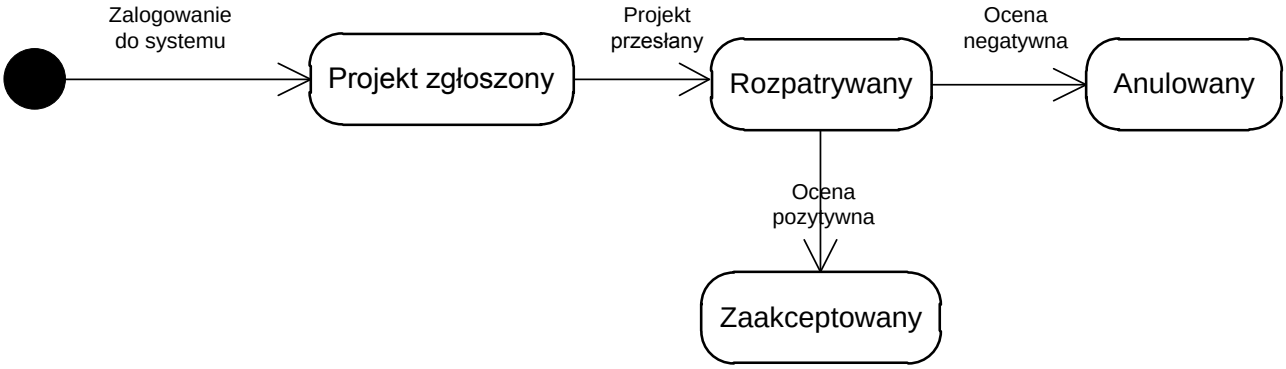
Rysunek 78. Projektowy diagram klas. *Źródło: Opracowanie własne.*

6.3. Diagramy stanów

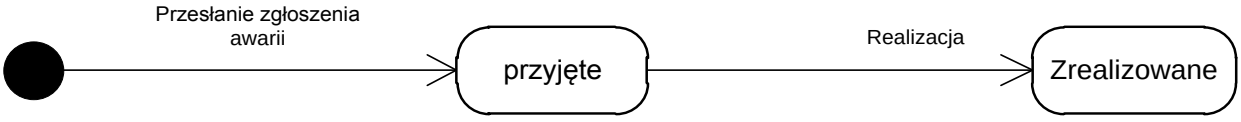
Diagram opisuje sekwencje stanów, w jakich kolejno przebywa dany obiekt. Stan obiektu rozumiany jest tutaj jako warunek określany przez jego atrybuty. Przykładowo – Rysunek 79 przedstawia obiekt klasy Użytkownik, którego można utworzyć, przypisać mu rolę, nadać uprawnienia oraz usunąć.



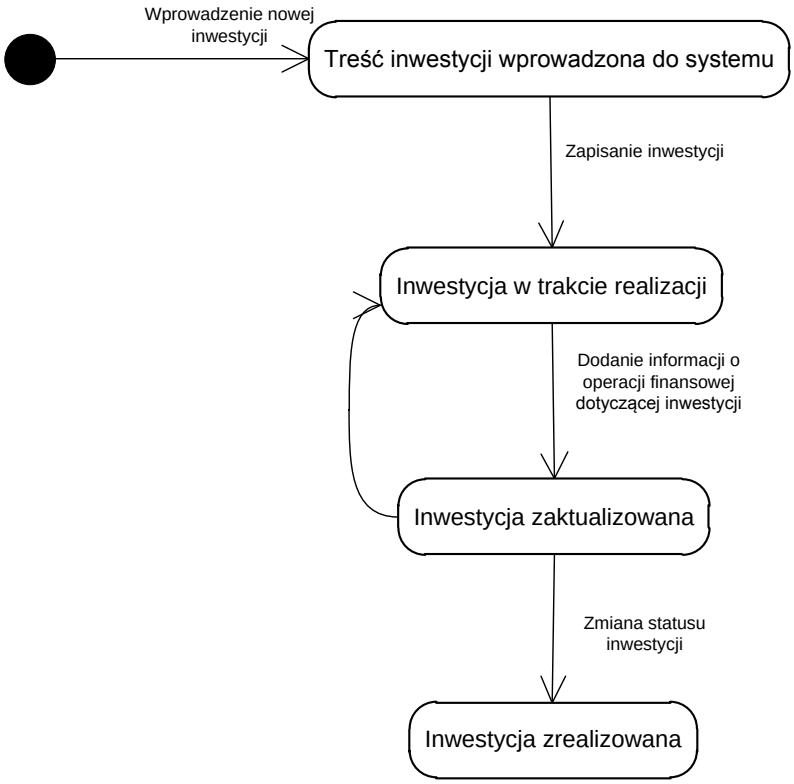
Rysunek 79. Diagram stanów Użytkownik. ***Źródło: Opracowanie własne.***



Rysunek 80. Diagram stanów Projekt. ***Źródło: Opracowanie własne.***



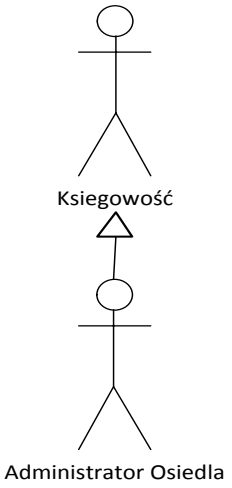
Rysunek 81. Diagram stanów Zgłoszenie awarii. ***Źródło: Opracowanie własne.***



Rysunek 82. Diagram stanów Inwestycja. ***Źródło: Opracowanie własne.***

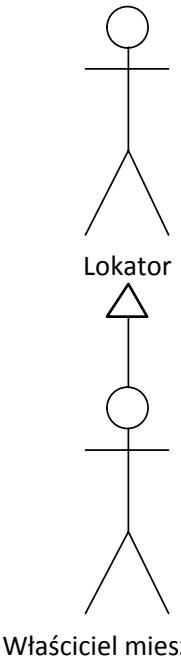
6.4. Diagram przypadków użycia

Diagram przypadków użycia reprezentuje funkcjonalności z perspektywy jego użytkowników oraz definiuje granicę modelowanego systemu. Wykorzystywany jest w fazie zbierania wymagań oraz koncentruje się na zachowaniu systemu postrzeganego z zewnątrz. Przypadek użycia opisuje funkcję realizowaną przez system jako efekt widoczny dla aktora. Przypadek wykonuje interakcje pomiędzy systemem a aktorem, który jest zewnętrzną encją. Diagram taki stanowi podstawę do testowania funkcji systemu w dalszych etapach projektu (**Błąd! Nie można odnaleźć źródła odwołania.**).

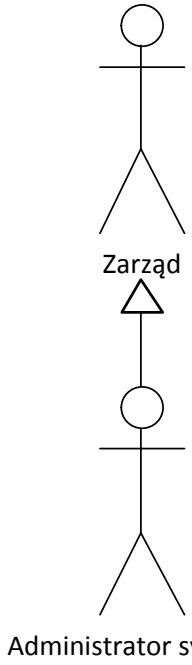


Rysunek 83. Dziedziczenie w diagramie przypadków użycia. ***Źródło: Opracowanie własne.***

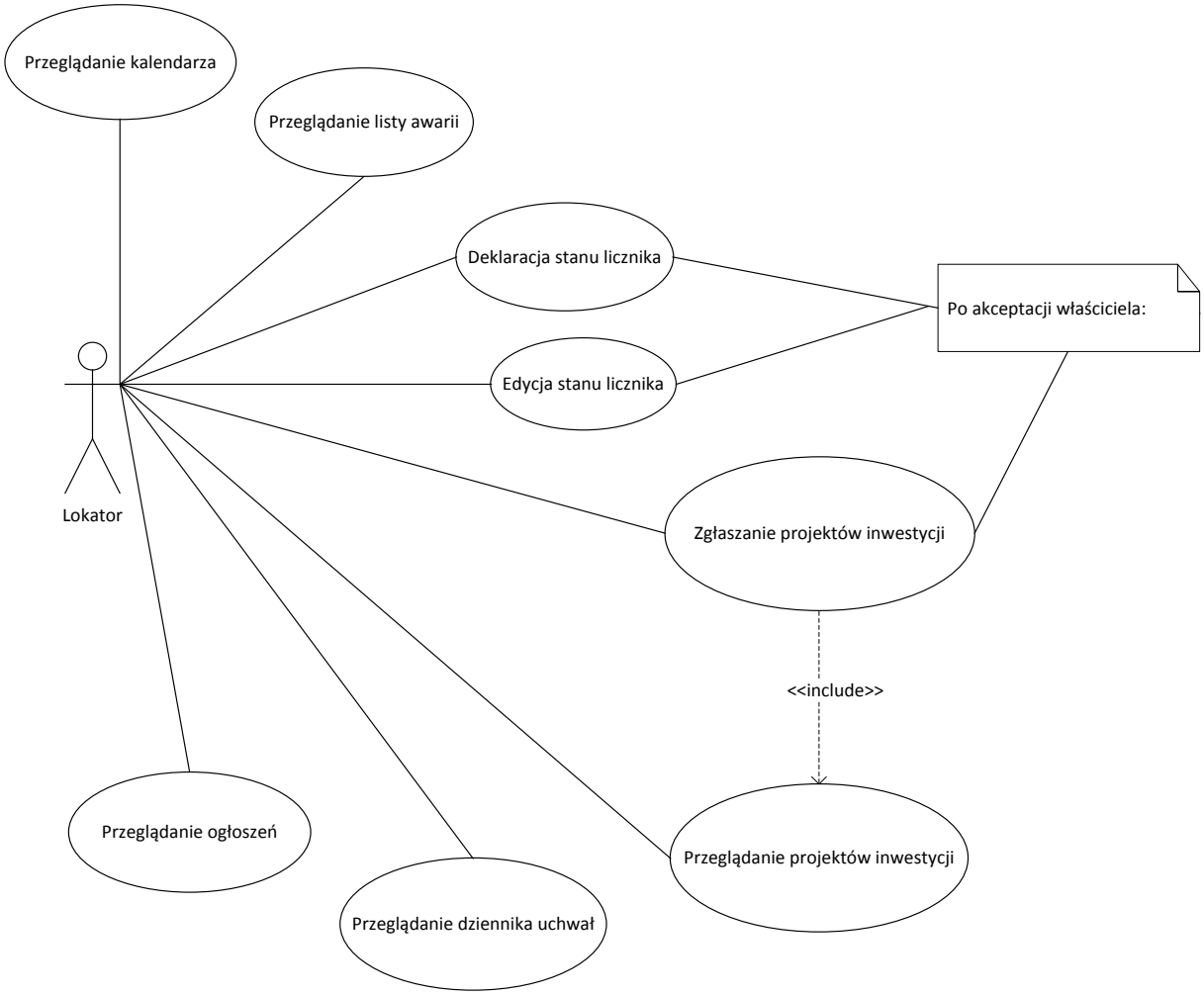
Diagram przypadków użycia został podzielony na części (Rysunek 86 do Rysunek 91) w celu lepszej czytelności. Rysunek 83, Rysunek 84, Rysunek 85 przedstawiają dziedziczenie funkcjonalności przez aktorów.



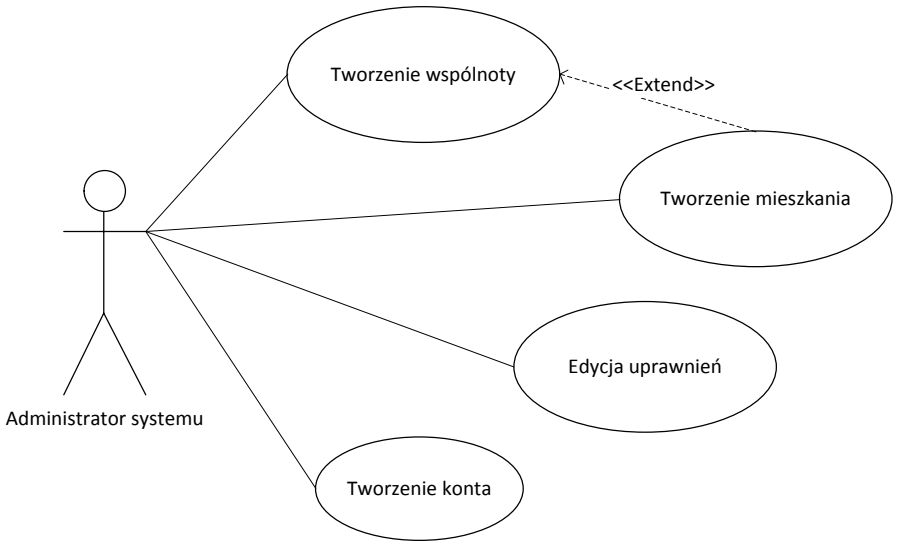
Rysunek 84. Dziedziczenie w diagramie przypadków użycia. ***Źródło: Opracowanie własne.***



Rysunek 85. Dziedziczenie w diagramie przypadków użycia. ***Źródło: Opracowanie własne.***



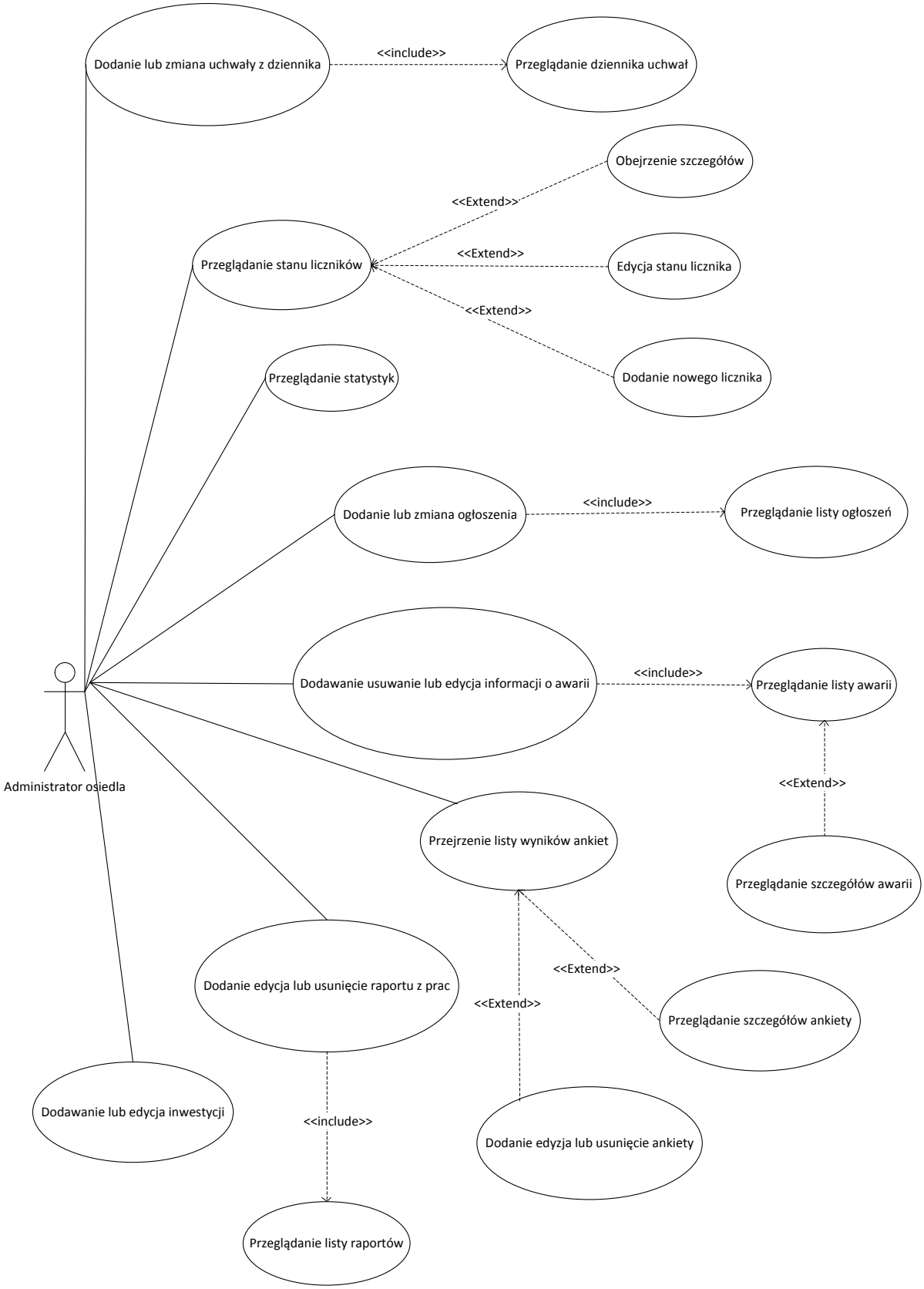
Rysunek 86. Diagram przypadków użycia Lokator. ***Źródło: Opracowanie własne.***



Rysunek 87. Diagram przypadków użycia Administrator Systemu. ***Źródło: Opracowanie własne.***

Rysunek 88. Diagram przypadków użycia Właściciel mieszkania. ***Źródło: Opracowanie własne.***

Rysunek 89. Diagram przypadków użycia Księgowość. ***Źródło: Opracowanie własne.***



Rysunek 90. Diagram przypadków użycia Administrator osiedla. ***Źródło: Opracowanie własne.***

Rysunek 91. Diagram przypadków użycia Zarząd. ***Źródło: Opracowanie własne.***

6.5. Scenariusze

Scenariusze przypadków użycia formułujemy w języku naturalnym, opierając się na terminach zamieszczonych w słowniku pojęć. Jest to konieczne z perspektywy komunikacji programistów z użytkownikami, którzy niekoniecznie znają terminologię i notację programistyczną. Opis każdego przypadku użycia jest specyfikacją przepływu zdarzeń między systemem a aktorem. W pracy scenariusze zostały przedstawione za pomocą tabel z głównym oraz alternatywnym przebiegiem zdarzeń. Załącznik 3 zawiera wszystkie scenariusze.

Tabela 3. Scenariusz przeglądania awarii. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Przeglądanie zgłoszeń awarii**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. W systemie istnieje co najmniej jedno publiczne zgłoszenie awarii.	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie zgłoszeń awarii w Menu
	2	System wyświetla wszystkie publiczne zgłoszenia posortowane malejąco względem daty
	3	Lokator wybiera zgłoszenie
	4	System wyświetla szczegóły wybranego zgłoszenia
	5	Lokator przegląda szczegóły zgłoszenia. Koniec przypadku użycia

Tabela 4. Scenariusz przeglądania projektów inwestycji. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Przeglądanie projektów inwestycji**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. W systemie istnieje co najmniej jeden projekt inwestycji	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie projektów inwestycji w Menu

	2	System wyświetla wszystkie projekty inwestycji
	3	Lokator wybiera projekt inwestycji
	4	System wyświetla szczegóły wybranego projektu inwestycji
	5	Lokator przegląda szczegóły projektu inwestycji. Koniec przypadku użycia

Tabela 5. Scenariusz zgłoszenia pomysłów inwestycji. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Zgłaszanie pomysłów na projekty inwestycji**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora.	
Warunki końcowe	Pomysł projektu inwestycji zostaje zaakceptowany przez właściciela mieszkania. Pomysł projektu zostaje poprawnie zapisany w systemie.	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie projektów inwestycji w Menu
	2	System wyświetla wszystkie projekty inwestycji
	3	Lokator wybiera dodanie nowego projektu inwestycji
	4	System wyświetla formularz dodawania nowego pomysłu projektu inwestycji
	5	Lokator wypełnia wszystkie wymagane pola w formularzu dodawania nowego projektu inwestycji
	6	Lokator wybiera zapisanie pomysłu projektu. Koniec PU

Tabela 6. Scenariusz zgłoszenia awarii. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Zgłaszanie awarii**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora	
Warunki końcowe	Zgłoszenie awarii zostaje poprawnie zapisane przez system	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**

	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie zgłoszeń awarii w Menu
	2	System wyświetla wszystkie projekty inwestycji
	3	Lokator wybiera dodanie nowego zgłoszenia
	4	System wyświetla formularz dodawania nowego zgłoszenia
	5	Lokator wypełnia wszystkie wymagane pola w formularzu dodawania nowego zgłoszenia i opcjonalnie wybiera zdjęcie nawiązujące do zgłoszenia
	6	Lokator wybiera zapisanie zgłoszenia. Koniec PU

6.6. Wymagania funkcjonalne

Zbieranie wymagań polega na komunikacji pomiędzy programistami, klientem i użytkownikami w celu wypracowania spójnej definicji nowego systemu. Błędy popełnione na etapie zbierania wymagań są bardzo kosztowne w usunięciu oraz dają o sobie znać zazwyczaj zaraz po przekazaniu systemu klientowi. Brak funkcjonalności może spowodować, że system będzie mało przydatny bądź, co gorsza, bezużyteczny. Wymagania funkcjonalne są opisem funkcjonalności, które projektowany system musi realizować i bez nich nie powinien istnieć. Tabela 7 przedstawia wymagania wraz z opisem oraz przypisem do aktorów, którzy powinni takie funkcjonalności posiadać.

Tabela 7. Wymagania funkcjonalne. ***Źródło: Opracowanie własne.***

L.p.	Nazwa wymagania	Opis wymagania	Ograniczenia	Aktorzy
1.1	Tworzenie wspólnot	Administrator wybiera Wyświetl wspólnoty z menu Panel administracyjny i przechodzi do widoku zarządzania wspólnotami z możliwością ich dodawania, usuwania i edytowania	Brak	Administrator systemu
1.2	Tworzenie mieszkań	Administrator wybiera Wyświetl mieszkania z menu Panel administracyjny i przechodzi do widoku zarządzania mieszkaniami z możliwością ich dodawania, usuwania i edytowania	Istnieje wspólnota oraz budynek/budynki, z którymi można powiązać mieszkania	Administrator systemu
1.3	Tworzenie kont	Administrator wybiera Zarządzanie użytkownikami z menu Panel administracyjny i przechodzi do widoku zarządzania użytkownikami z możliwością ich dodawania, usuwania i edytowania	Brak	Administrator systemu
1.4	Edycja	Administrator wybiera Zarządzanie	Istnieje baza	Administrator

	uprawnień	użytkownikami z menu Panel administracyjny i przechodzi do widoku zarządzania użytkownikami, wybiera użytkownika, którego chce edytować, po czym system wyświetla mu formularz edycji danych użytkownika (w tym uprawnień)	kont w aplikacji	systemu
2.2	Przeglądanie zgłoszeń	Użytkownik wybiera Przeglądaj awarię z menu Wspólnota, po czym system wyświetla listę najnowszych awarii zaistniałych w danej wspólnocie	Istnieją zgłoszenia powiązane ze wspólnotą, w której pracuje zarządca	Zarządca, lokator, właściciel, zarząd
2.3	Zmiana statusu zgłoszeń	Zarządca wyświetla listę awarii klikając w Przeglądaj awarię z menu Wspólnota, wybiera awarię, którą chce edytować, po czym z pozycji Status na formularzu wybiera aktualny status awarii.	Istnieje co najmniej jedno zgłoszenie w bazie wspólnoty	Zarządca
2.4	Wstawianie ogłoszeń	Użytkownik wybiera Wyświetl ogłoszenia z menu Wspólnota, po czym na liście wspólnot klika w ikonkę plusika, a następnie system wyświetla formularz dodawania ogłoszenia	Brak	Zarządca, Zarząd
3.1	Odrzucanie pomysłu inwestycji	Przedstawiciel zarządu z menu Inwestycje wybiera przycisk Przeglądaj projekty, i z listy projektów klika w przycisk Odrzuć dla projektu, który chce odrzucić	Istnieje co najmniej jeden nierozpatrzony pomysł inwestycji	Zarząd
3.2	Odrzucanie pomysłu inwestycji do poprawy	Przedstawiciel zarządu z menu Inwestycje wybiera przycisk Przeglądaj projekty i z listy projektów klika w przycisk Odrzuć do poprawy dla projektu, który chce odrzucić	Istnieje co najmniej jeden nierozpatrzony pomysł inwestycji	Zarząd
3.3	Akceptacja pomysłu inwestycji	Przedstawiciel zarządu z menu Inwestycje wybiera przycisk Przeglądaj projekty i z listy projektów klika w przycisk Zaakceptuj dla projektu, który chce odrzucić	Istnieje co najmniej jeden nierozpatrzony pomysł inwestycji	Zarząd
3.4	Wprowadzanie zdarzeń do kalendarza	Przedstawiciel zarządu wybiera przycisk Kalendarz z głównego menu, po czym wyświetla mu się widok kalendarza, na którym może umieszczać zdarzenia w ramach konkretnego dnia	Brak	Zarząd
3.5	Wprowadzanie uchwał do	Przedstawiciel zarządu wybiera Lista uchwał z menu Wspólnota, system wyświetla widok z	Brak	Zarząd

	dziennika	zestawem wszystkich uchwał w bazie wspólnoty, po czym użytkownik wybiera uchwałę, którą chce edytować, zaś system wyświetla formularz edycji parametrów ustawy, w tym możliwość dodania załącznika w postaci pliku		
3.6	Wprowadzanie i edytowanie informacji o inwestycjach	Przedstawiciel zarządu z menu Inwestycje wybiera przycisk Przeglądaj inwestycje, system wyświetla zbiór wszystkich aktywnych inwestycji w ramach wspólnoty, zaś użytkownik klikając w ikonkę ołówka przy pożądanej inwestycji uruchamia formularz edycji informacji o inwestycjach	Brak	Zarząd
3.7	Przeglądanie statystyk zużycia (liczników) oraz opłat	Przedstawiciel zarządu z menu Wspólnota wybiera przycisk Statystyki, zaś system wyświetla zagregowane informacje o zużyciu oraz opłatach za ostatni tydzień i miesiąc w ramach danej wspólnoty	Brak	Zarząd
4.1	Wprowadzanie wpłat	Księgowy wybiera Lista płatności z menu Wspólnota, a następnie ikonkę plusika na liście płatności, zaś system wyświetla formularz dodawania informacji o dokonanej płatności	Brak	Księgowość
4.2	Akceptacja deklaracji stanów liczników	Księgowy z menu Panel administracyjny wybiera Przeglądaj liczniki, po czym system wyświetla listę wszystkich liczników. Użytkownik może wyświetlić listę stanów konkretnego licznika, klikając na przycisk Stany licznika przy wybranym liczniku, po czym na liście stanów może kliknąć Zaakceptuj dla jednego z niezaakceptowanych stanów danego licznika	Są zadeklarowane informacje o stanie liczników mieszkańców	Księgowość
5.1	Przeglądanie inwestycji	Użytkownik z menu Inwestycje wybiera przycisk Przeglądaj inwestycje, system wyświetla zbiór wszystkich aktywnych inwestycji w ramach wspólnoty	Istnieje co najmniej jedna inwestycja w bazie danej wspólnoty	Lokator, właściciel, zarząd, zarządca
5.2	Przeglądanie dziennika uchwał	Użytkownik z menu Wspólnota wybiera przycisk Lista uchwał, a system wyświetla dziennik uchwał – listę dotychczasowych uchwał w ramach wspólnoty	Brak	Lokator, właściciel, zarząd, zarządca
5.3	Przeglądanie ogłoszeń	Użytkownik z menu Wspólnota wybiera przycisk Wyświetl ogłoszenia, zaś system wyświetla zbiór wszystkich aktualnych ogłoszeń	Istnieje co najmniej jedno ogłoszenie w ramach	Lokator, właściciel, zarząd,

		opublikowanych w ramach wspólnoty	wspólnoty	zarządca
5.4	Deklaracja i edycja stanu licznika	Użytkownik z menu Panel administracyjny wybiera Przeglądaj liczniki, po czym system wyświetla listę wszystkich liczników. Użytkownik może wyświetlić listę stanów konkretnego licznika, klikając na przycisk Stany licznika przy wybranym liczniku, po czym korzystając z przycisku Edytuj, może wprowadzić zmiany w stanie konkretnego licznika	Brak	Lokator, właściciel
5.5	Zgłaszanie awarii	Użytkownik z menu Wspólnota wybiera przycisk Wyświetl awarie, po czym na liście wszystkich awarii klika przycisk dodawania nowej awarii, uruchamiając stosowny formularz	Brak	Lokator, właściciel
5.6	Zgłaszanie pomysłów na inwestycje	Użytkownik z menu „Inwestycje” wybiera przycisk „Przeglądaj projekty”, po czym w widoku wszystkich projektów korzysta z przycisku dodawania nowego projektu inwestycji uruchamiając tym samym stosowny formularz	W przypadku lokatora wymaga akceptacji właściciela jego mieszkania	Lokator, właściciel
6.1	Akceptacja zgłoszeń lokatora	Użytkownik z menu Inwestycje wybiera przycisk Przeglądaj projekty, po czym w widoku wszystkich projektów wybiera przycisk Zaakceptuj przy jednym z projektów zgłoszonych przez lokatora, które do tej pory nie zostały jeszcze rozpatrzone przez właściciela	Lokator dokonał zgłoszenia	Właściciel
6.2	Edycja deklaracji osób	Właściciel z menu Panel administracyjny wyświetla listę mieszkań, klikając w przycisk Wyświetl mieszkania. Z listy mieszkań użytkownik wybiera przycisk edycji dla wybranego mieszkania, zaś system wyświetla formularz edycji informacji o mieszkaniu, w tym liczbę zamieszkałych tam osób	Brak	Właściciel
6.3	Głosowanie w ankietach	Właściciel z menu Wspólnota wybiera przycisk Wyświetl ankiety, zaś system wyświetla listę wszystkich aktywnych ankiet. Użytkownik wybiera pożądaną ankietę, zaś w nowym oknie wybiera przycisk Zagłosuj, uruchamiając tym samym okienko z możliwością wyboru odpowiedzi na pytanie z ankiety	Została zainicjowana co najmniej jedna aktywna ankieta w ramach wspólnoty	Właściciel
6.4	Przeglądanie kalendarza	Właściciel zarządu wybiera przycisk Kalendarz z głównego menu, po czym wyświetla mu się	Brak	Właściciel

	wspólnoty	widok kalendarza, na którym umieszczone są zdarzenia w ramach konkretnego dnia		

6.7. Wymaganie niefunkcjonalne

Wymaganie niefunkcjonalne opisują aspekty systemu, które nie są bezpośrednio związane z jego funkcjonalnością. Dotyczą zazwyczaj cech użyteczności, niezawodności, wydajności, bezpieczeństwa oraz wspieralności, wyrażającej łatwość wprowadzania zmian po jego wdrożeniu. Zestawienie takich funkcjonalności prezentuje Tabela 8.

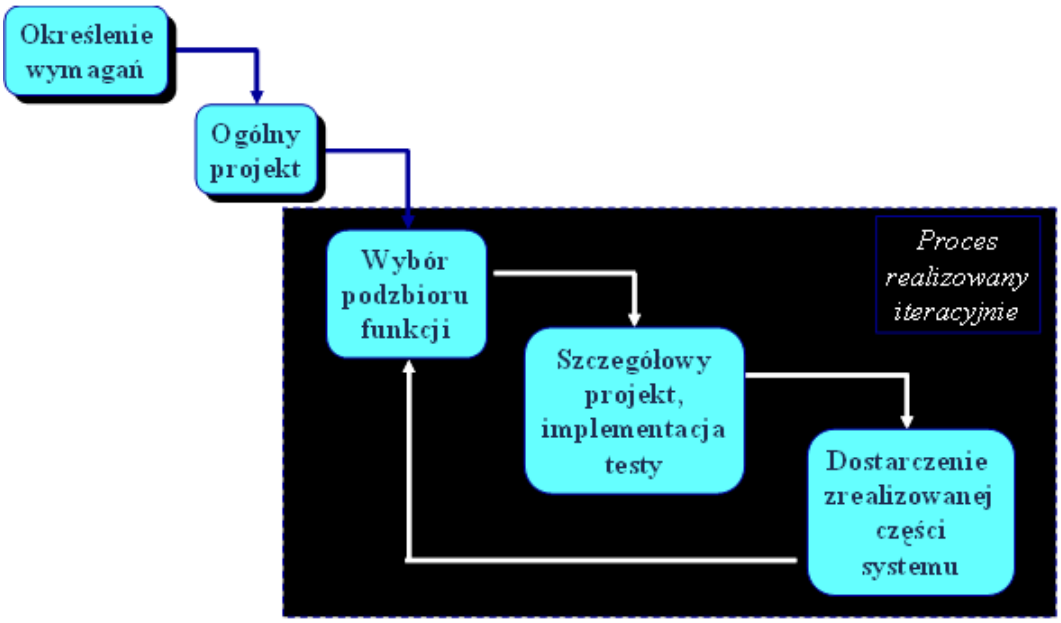
Tabela 8. Wymagania niefunkcjonalne. ***Źródło: Opracowanie własne.***

Lp.	Cecha	Priorytet	Opis
1	Wydajność	Musi	Minimalna ilość klientów korzystających z portalu w jednym czasie to 150.
2	Wydajność	Musi	Czas przekierowania użytkownika system z panelu logowania do funkcjonalności nie dłuższy niż 2 s.
3	Bezpieczeństwo	Musi	Generowanie kopii bezpieczeństwa.
4	Bezpieczeństwo	Musi	Musi wykorzystywać bezpieczne mechanizmy szyfrowania przesyłanych danych (https).
5	Bezpieczeństwo	Musi	Dostęp do danych po wpisaniu hasła i nazwy użytkownika.
6	Bezpieczeństwo	Musi	Dane klientów i wspólnoty zabezpieczone przed dostępem osób postronnych.
7	Niezawodność	Powinien	System w godzinach od 8 do 16 powinien być dostępny przez 99% czasu.
8	Niezawodność	Powinien	System w godzinach od 16 do 8 powinien być dostępny przez 95% czasu.
9	Obsługa	Musi	Musi być w 100% kompatybilna z przynajmniej jedną z wymienionych przeglądarek internetowych: Internet Explorer 11, Opera 28, Chrome 42, Firefox 37.
10	Obsługa	Powinien	System powinien pracować na systemach operacyjnych Windows XP Vista 7,8,10.
11	Obsługa	Musi	System musi pracować w architekturze wielowarstwowej uruchamianej poprzez przeglądarkę internetową w oparciu o serwer aplikacyjny.
12	Kodowanie	Musi	System musi być wykonany w technologii Java.
13	Kodowanie	Musi	Baza danych kompatybilna z językiem SQL.
14	Kodowanie	Musi	Możliwość łatwej rozbudowy, np. wersji mobilnej.

15	Użyteczność	Powinien	Optymalne wyświetlanie na rozdzielczości 1366x768.
16	Użyteczność	Powinien	Powinna być zapewniona responsywność strony dla wielu urządzeń.

6.8. Implementacja

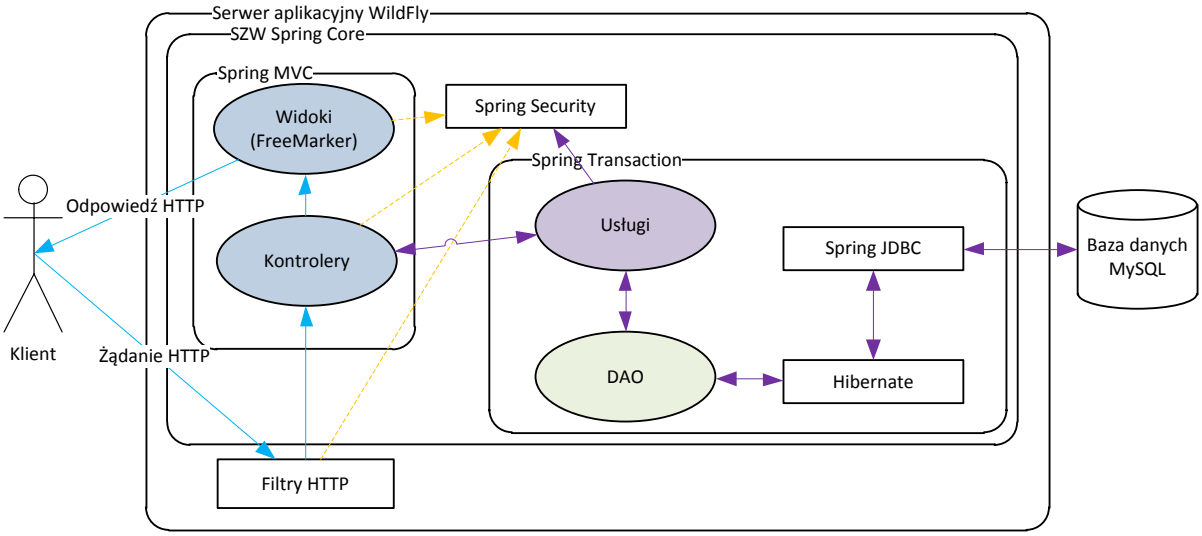
Cykl życia oprogramowania, który wykorzystaliśmy w naszej aplikacji, to model przyrostowy (Rysunek 92). Pozwolił nam na określenie wymagań oraz wykonanie szkieletu systemu. Takie podejście pozwoliło także na sukcesywne dostarczanie funkcjonalności przez zespół implementacji. Prace zostały podzielone na kilka faz. Każda iteracja zawierała opis funkcjonalności, swój priorytet oraz przypis do osoby, której zadaniem była implementacja. Dzięki temu, już po pierwszym etapie zespół testów na bieżąco mógł zgłaszać błędy i dbać o jakość wytwarzanego oprogramowania. Dzięki realizacji przyrostowej, już po pierwszym etapie mieliśmy widoczne efekty naszej pracy.



Rysunek 92. Model Przyrostowy. *Źródło: [37].*

6.8.1 Architektura

Szkielet aplikacji został wykonany na podstawie Spring Core, dzięki czemu zarządzanie cyklem życia obiektów biznesowych oraz wstrzykiwanie zależności między nimi zostało zautomatyzowane i przejęte przez framework. W aplikacji wykorzystaliśmy konwencję konfiguracji poprzez adnotacje języka Java (JavaConfig), dzięki czemu cała aplikacja, z zastrzeżeniem widoków, korzysta wyłącznie z języka Java, bez dodatkowych plików XML. Przyspieszyło to rozwój aplikacji, ponieważ nie było potrzeby dodawać definicji obiektów zarządzanych przez Spring nigdzie poza plikiem klasy definiującej dany obiekt. W celu ułatwienia wstrzykiwania zależności wszędzie stosujemy adnotację @Autowired, sprowadzając identyfikację danego zależnego obiektu do sprawdzenia klasy, do której należy - z wyjątkiem kilku obiektów dostarczających implementacji interfejsów ze Spring Security. Nie odwoływaliśmy się do beanów springowych po nazwie, a jedynie po klasie. Rysunek 93 wizualnie przedstawia architekturę aplikacji.



Rysunek 93. Architektura aplikacji. ***Źródło: Opracowanie własne.***

Definicja konfiguracji aplikacji zaczyna się w deskryptorze aplikacji webowej, plik web.xml, poniżej (Listing 4) jego fragment odpowiadający za uruchamianie kontekstu Spring.

Listing 4. Web.xml. ***Źródło: Opracowanie własne.***

```
<!-- klasa gł. kontekstu Spring-Wymusza konfigurację typu JavaConfig(@Configuration) -->
<context-param>
	<param-name>contextClass</param-name>
	<param-value>org.springframework.web.context.support.
		AnnotationConfigWebApplicationContext</param-value>
</context-param>
<!--klasa konfiguracji głównego kontekstu Spring -->
<context-param>
	<param-name>contextConfigLocation</param-name>
	<param-value>pl.edu.pja.io2014.szw.core.config.AppConfig</param-value>
</context-param>
<!--klasa inicjująca utworzenie gł. kontekstu Spring napodstawie w.w. konfiguracji -->
<listener>
	<listener-class>org.springframework.web
		.context.ContextLoaderListener</listener-class>
</listener>
```

Ładowana klasa AppConfig konfiguracji kontekstu importuje konfigurację SecurityConfig, zawierającą konfigurację Spring Security. Ta z kolei importuje konfigurację CoreConfig, która definiuje kontekst startowy aplikacji. Listing 5 przedstawia fragment CoreConfig.java.

Listing 5. CoreConfig.java. ***Źródło: Opracowanie własne.***

```
@Configuration
@EnableTransactionManagement // Włącza zarządzanie transakcjami
/* Włącza tworzenie Proxy między beanami o różnym zakresie. Parametr określa,
że do tworzenia Proxy będzie wykorzystywany mechanizm CGLib, a nie JDK Proxy.*/
@EnableAspectJAutoProxy(proxyTargetClass = true)
// ładuje plik properties,wartości z niego mogą być wykorzystane w adnotacji @Value
@PropertySource("classpath:szw.properties")
// Pakiety skanowane w celu wyszukania beanów Springowych
@ComponentScan({"pl.edu.pja.io2014.szw.core","pl.edu.pja.io2014.szw.model"})
```

```
public class CoreConfig {
    @Autowired
    private AppProperties appProperties;
    @Bean
    public LocalSessionFactoryBean getSessionFactory() {…
    }
    @Bean
    public DataSource getDataSource() {…
    }
    @Bean
    @Autowired
    public HibernateTransactionManager transactionManager(SessionFactory sessionFactory
    ) {…
    }
    @Bean
    @Autowired
    public HibernateTemplate getHibernateTemplate(SessionFactory sessionFactory
    ) {…
    }
}
```

W wykonanej aplikacji widoczne są następujące warstwy:

Mapowanie obiektowo-relacyjne – klasy występujące w diagramie projektowym zostały zdefiniowane jako zwykłe obiekty Javowe, jednak z dodatkowymi adnotacjami, które Hibernate odpowiednio obsługuje. Przykładowe użyte adnotacje:

- @Entity - oznacza tabelę (encję). Deklaracja klasy z tą adnotacją powoduje potraktowanie klasy jako zmapowaną encję, tzn. można wykonywać na niej operacje CRUD.
- @Table - w przypadku Hibernate służy do przekazania dodatkowych informacji o realizacji encji - ponieważ wykorzystujemy bazę MySQL, adnotacja ta służyła do wskazywania nazwy tabeli
- @Id - określa klucz główny encji
- @MappedSuperclass - pozwala na realizację dziedziczenia w warstwie bazodanowej, klasy oznaczone tą adnotacją nie są bezpośrednio realizowane jako tabela, a jedynie jej podklasy.
- @Embeddable i @EmbeddedId - dwie powiązane ze sobą adnotacje pozwalające ponownie wykorzystać definicję klucza głównego dla różnych encji. @Embeddable oznacza klasę, której treść jest kluczem głównym, @EmbeddedId oznacza atrybut encji, który należy do klasy oznaczonej @Embeddable i będzie traktowany - jako cały - jako klucz główny encji.
- adnotacje dla relacji - @OneToMany, @ManyToOne, @OneToOne, @ManyToMany - służą do definiowania relacji między encjami, ich nazwy bezpośrednio wskazują na charakter relacji.
- @Column - pozwala przekazać dodatkowe informacje o kolumnie reprezentującej atrybut (sposób realizacji atrybutu, długość, nazwa, itp.)
- @JoinColumn - dla relacji @ManyToOne i @OneToOne pozwala przekazać takie informacje jak @Column dla zwykłego atrybutu
- @JoinTable - dla relacji @ManyToMany pozwala określić definicję tabeli asocjacyjnej między encjami
- @Version - wskazuje na kolumnę "wersjonującą" obiekt, pozwalającą zrealizować optymistyczne blokowanie wiersza tabeli.

W aplikacji przyjęto, że wszystkie encje "biznesowe" są wersjonowane. Wszystkie obiekty posiadają również flagę wskazującą, czy obiekt jest aktywny, czy nie. Dzięki temu możliwe jest przechowywanie historycznych wpisów. Klasa bazowa wszystkich encji została przedstawiona na

Listing 6.

Listing 6. Klasa bazowa. *Źródło: Opracowanie własne.*

```
@MappedSuperclass
public abstract class AbstractEntity <K extends Serializable> implements Serializable {

        @Version
        @Column(name = "version", nullable = false)
        private Long version;
        @Column(name = "disabled", nullable = false)
        @Type(type = "yes_no")
        private Boolean disabled;
        @Column(name = "created", nullable = false, updatable = false)
        private Timestamp created;
        @Column(name = "modified", nullable = false)
        private Timestamp lastModified;

        protected AbstractEntity() {
             disabled = false;
        }

        public abstract K getId();
        public abstract void setId(K id);
}
```

Warstwa DAO – obiektów dostępu do danych. Tylko w tych obiektach dostępne są odwołania do komponentów mogących wykonywać faktyczne operacje na bazie. Do mapowania obiektowo-relacyjnego, konstrukcji zapytań i zarządzania transakcjami została wykorzystana biblioteka Hibernate. Ze względu na potrzebę integracji ze Spring, został wykorzystany moduł Spring ORM. Moduł ten dostarcza obiekt klasy HibernateTemplate, który umożliwia delegację wszystkich operacji bazodanowych do jednego obiektu z automatycznym uwzględnieniem transakcyjności. Klasy DAO (Listing 7) udostępniają operacje odczytu, tworzenia, aktualizacji obiektów na bazie. Usuwanie obiektów jest realizowane poprzez oznaczenie obiektu jako nieaktywny. Odczyt danych jest możliwy przy użyciu 2 mechanizmów:

- Criteria API - w celu stworzenia zapytania przekazywane są do DAO ograniczenia zakładane na zapytanie (klasa Criterion, odpowiadająca elementowi klauzuli "WHERE"), rzutowanie (klasa Projection, odpowiadająca liście kolumn w "SELECT" i ewentualnym agregacjom), sposób uporządkowania (klasa Order, odpowiadająca "ORDER BY"). Zapytania w takim wypadku są konstruowane i analizowane na bieżąco. Nie jest to wydajne rozwiązanie dla złożonych zapytań, jednak jest wykorzystywane w przypadku, jeśli nie można zawczasu określić kształtu zapytania, tak jest w przypadku przekazywania danych zależnych od użytkownika, np. ograniczenia zwracanej listy do obiektów, do których użytkownik ma uprawnienia lub wprowadzenia kryteriów wyszukiwania przez użytkownika.
- nazwane zapytania (wersje Hibernate Query Language - HQL - oraz natywne) - wykorzystywane w przypadku, jeśli metoda DAO realizuje bardzo konkretną funkcję i kształt zapytania jest zawczasu znany, np. dla statystyk.

Listing 7. Klasa bazowa DAO. *Źródło: Opracowanie własne.*

```
@Repository
@Transactional
public abstract class AbstractDAO<K extends Serializable, E extends AbstractEntity<K>> {
    @Autowired
    protected HibernateTemplate hibernateTemplate;
    public abstract Class<E> getEntityClass();
    public List<E> list(Criterion... criteria) {
```

```
        return list(null, null, null, criteria);
    }
    @SuppressWarnings("unchecked")
    public List<E> list(Integer offset, Integer limit, Order[] order,
         Criterion... criteria) {
        return ((List<E>) getBaseQuery(offset, limit, order, criteria)
                    .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY).list()).stream()
                    .peek(this::initialize).collect(Collectors.<E>toList());
    }
    public List<?> listWithProjection(Projection projection, Integer offset,
                    Integer limit, Order[] order, Criterion... criteria) {
        return getProjectionQuery(projection, offset, limit, order, criteria).list();
    }
    private Criteria getProjectionQuery(Projection projection, Integer offset,
                    Integer limit, Order[] order, Criterion[] criteria) {
        return getBaseQuery(offset, limit, order, criteria).setProjection(projection);
    }
    public Criteria getBaseQuery(Integer offset, Integer limit, Order[] orders,
         Criterion... criteria) {
        Criteria query = hibernateTemplate
                .getSessionFactory()
                .getCurrentSession()
                .createCriteria(getEntityClass())
                .add(Restrictions.eq("disabled", Boolean.FALSE));
        if (criteria != null) {
            for (Criterion criterion : criteria) {
                query.add(criterion);
            }
        }
        if (orders != null) {
            for (Order order : orders) {
                query.addOrder(order);
            }
        }
        if (offset != null && limit != null) {
            query.setFirstResult(offset).setMaxResults(limit);
        }
        returnquery;
    }
    public long count(Criterion... criteria) {
        return (long) getProjectionQuery(Projections.rowCount(), null, null, null,
                        criteria).uniqueResult();
    }
    public E getById(K id) {
        return hibernateTemplate.load(getEntityClass(), id);
    }
    public E getByIdAndVersion(K id, longversion) {
        return load(id, version);
    }
    public E load(K id, longversion) {
        E result = hibernateTemplate.load(getEntityClass(), id);
        if (result != null && version != result.getVersion()) {
            fireStaleObjectException(id);
        }
        return result;
    }
    public K save(E entity) {
        return saveInternal(entity);
```

```
    }
    public void disable(E entity) {
        entity.setDisabled(true);
        saveInternal(entity);
    }
    protected K saveInternal(E entity) {
        hibernateTemplate.saveOrUpdate(entity);
        return entity.getId();
    }
    public void initialize(E entity) {
        hibernateTemplate.initialize(entity);
    }
    public void initializeAll(E entity) {
        initialize(entity);
    }
}
```

Warstwa usług - stanowi pomost między DAO i kontrolerami i odpowiada za udostępnienie logiki biznesowej przy pomocy metod dostarczanych przez obiekty DAO. Metody usług zbierają w logiczny i spójny ciąg sekwencję operacji bazodanowych zmierzających do wykonania określonej operacji biznesowej. W tej warstwie również wykorzystujemy zarządzanie transakcjami, przy czym transakcje są propagowane do DAO, tzn. w przypadku odwołań do kilku DAO lub wykonania kilku zapytań w jednej metodzie warstwy usług, wszystkie te zapytania są wykonywane w obrębie wspólnej transakcji. W aplikacji wykorzystano tę warstwę również do przygotowania danych testowych i startowej konfiguracji aplikacji (nie wykorzystujemy zwykłych zapytań w SQL ani HQL, wszystkie dane testowe są mapowane z obiektów javowych). Listing 8 przedstawia klasę bazową usług.

Listing 8. Klasa bazowa usług. ***Źródło: Opracowanie własne.***

```
@Transactional
public abstract class AbstractEntityService<K extends Serializable,
       E extends AbstractEntity<K>, D extends AbstractDAO<K, E>> {
    protected final D dao;
    protected AbstractEntityService(D dao) {
        this.dao = dao;
    }
    public List<E> list(Criterion... criteria) {
        return list((Order[]) null, criteria);
    }
    public List<E> list(Order order, Criterion... criteria) {
        return list(null, null, order, criteria);
    }
    public List<E> list(Order[] orders, Criterion... criteria) {
        return list(null, null, orders, criteria);
    }
    public List<E> list(intoffset, intlimit) {
        return list(offset, limit, (Order[]) null);
    }
    public List<E> list(Integer offset, Integer limit, Order[] orders, Criterion
        … criteria) {
    return dao.list(offset, limit, orders, criteria);
    }
    public List<E> list(Integer offset, Integer limit, Order order, Criterion
        … criteria){
    return list(offset, limit, new Order[]{order}, criteria);
    }
    public long count(Criterion... criteria) {
        return dao.count(criteria);
```

```
    }
    public E getInstance(K id) {
        final E entity = dao.getById(id);
        dao.initialize(entity);
        return entity;
    }
    public E getInstance(K id, Long version) {
        E result = version == null ? dao.getById(id) : dao.getByIdAndVersion(id, version);
        dao.initialize(result);
        return result;
    }
    public E getInstanceFull(K id, Long version) {
        E result = version == null ? dao.getById(id) : dao.getByIdAndVersion(id, version);
        dao.initializeAll(result);
        return result;
    }
    public void delete(K id) {
        E entity = dao.getById(id);
        dao.disable(entity);
    }
    public void delete(E entity) {
        dao.disable(entity);
    }
    public K save(E entity) {
        return dao.save(entity);
    }
    public void createTestData() {
    }
}
```

Warstwa kontrolerów - odpowiada za realizację żądań klienta przy pomocy metod udostępnianych przez warstwę usług. Metody kontrolerów nie są transakcyjne. Do realizacji kontrolerów została wykorzystana biblioteka Spring MVC i podobnie, jak w przypadku samego Spring Core, cała konfiguracja jest zrealizowana przy pomocy adnotacji. Wykorzystywane adnotacje:

- @Controller - identyfikuje klasę kontrolera
- @RequestMapping - określa, jaki URL i jaką metodę żądania obsługuje dana metoda kontrolera.
- @PathVariable - określa parametr wywołania metody kontrolera, którego wartość jest automatycznie przez Spring MVC wyliczana i udostępniana w metocdzie na podstawie URL, umożliwia to realizację REST-owej obsługi żądań.
- @ModelAttribute - określa, że dany parametr metody jest dostępny w parametrach żądania jako jeden atrybut. W przypadku metod "POST" do konstrukcji obiektu wykorzystywany jest mechanizm introspekcji. W przypadku metod "GET" jest to obiekt dostarczony przez kontroler. Obiekt taki jest dostępny w widokach.
- @ResponseBody - określa, że zwracana wartość to treść odpowiedzi. Wykorzystywana przy operacjach AJAX, konkretnie dla ekranu kalendarza.
- @ControllerAdvice - umożliwia, przy pomocy mechanizmu AOP Springa, zdefiniowanie wspólnego kodu dla wszystkich kontrolerów, co jest wykorzystywane w połączeniu z następną adnotacją:
- @ExceptionHandler - adnotacja określa metodę, która obsługuje dany błąd (wyjątek Java, status HTTP, itp.). Wykorzystywana do generowania widoków błędów.
- @PreAuthorize - adnotacja Spring Security umożliwiająca ograniczenie dostępu do metody do użytkownika o odpowiednich uprawnieniach. W przypadku niewystarczających uprawnień generowany jest widok dla statusu HTTP 403.

Kontrolery mogą zwracać identyfikator widoku (String), sam widok (View), widok i model łącznie (ModelAndView) lub ciało odpowiedzi (String, ale z dodatkową adnotacją metody @ResponseBody). W aplikacji wykorzystywaliśmy prawie wyłącznie pierwsze podejście oraz @ResponseBody.

Warstwa widoków odpowiada za generowanie odpowiedzi do klienta, wykonana za pomocą szablonów FreeMarker oraz dodatkowych bibliotek JS (dhtmlxplanner, jQuery) oraz CSS (Bootstrap).

Powyższy opis prezentuje "poziomy" podział aplikacji. W celu organizacji i wprowadzenia podziału funkcjonalnego, przyjęto, że każda główna encja biznesowa ma swoje dedykowane DAO i dedykowaną usługę. Dzięki temu możliwe było

- duże ponowne użycie kodu, ponieważ metody DAO i usług bardzo często miały identyczne wywołania
- czytelny podział odpowiedzialności usług - wyszukiwanie wspólnot jest w serwisie dla wspólnot, itp.
- podział pracy - różne osoby mogły się zająć przekrojowo implementacją dla różnych encji.

Fragment przykładowego kontrolera, kontroler dla kalendarza, został przedstawiony na Listing 9.

Listing 9. Kontroler kalendarza. ***Źródło: Opracowanie własne.***

```
@Controller
@RequestMapping("/secure/calendar")
public class CalendarController {
    private final EventService eventService;
    private final HomeownerAssociationService homeownerAssociationService;

    @Autowired
    public CalendarController(EventService eventService, HomeownerAssociationService
          homeownerAssociationService) {…
        }

    @RequestMapping(method = RequestMethod.GET)
    @PreAuthorize("hasRole('MEETING_READ')")
    public String plannerRedirect(Authentication authentication) throws Exception {
        List<Long> homeownerAssociationIdsForUser =
        homeownerAssociationService.getUserAssociationIds(authentication,
        UserPermission.MEETING_READ);
        return "redirect:/web/secure/calendar/"
        +homeownerAssociationIdsForUser.iterator().next();
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @PreAuthorize("hasRole('MEETING_READ')")
    public String planner(Authentication authentication,
            @PathVariable Long id,
            @ModelAttribute("model") ModelMap modelMap,
            RedirectAttributes redirectAttributes) throws Exception {
       List<Long> homeownerAssociationIdsForUser
        =homeownerAssociationService.getUserAssociationIds(authentication,
        UserPermission.MEETING_READ);
        modelMap.addAttribute("body", createPlannerBody(authentication, id));
        modelMap.addAttribute("homeownerAssociations",homeownerAssociationService.list
             (Restrictions.in("id",homeownerAssociationIdsForUser)));
        modelMap.addAttribute("selectedAssociationId", id);
        return "calendar/list";
    }
    @RequestMapping(value = "/{id}", method = RequestMethod.POST)
```

```
    @PreAuthorize("hasRole('MEETING_READ')")
    public String changeAssociation(Authentication authentication, @PathVariable Long id,
            @ModelAttribute("homeownerAssociationId") Long homeownerAssociationId,
            RedirectAttributes redirectAttributes) {
       List<Long> homeownerAssociationIdsForUser
        =homeownerAssociationService.getUserAssociationIds(authentication,
        UserPermission.MEETING_READ);
        if (homeownerAssociationIdsForUser.contains(homeownerAssociationId)) {
            return "redirect:/web/secure/calendar/" + homeownerAssociationId;
        } else {
            return "redirect:/web/secure/calendar/";
        }
    }

    private String createPlannerBody(Authentication authentication, Long id) throws
            Exception {…
        return p.render();
    }

    @RequestMapping(value = "/events/{id}", produces = "application/json; charset=utf-8")
    @ResponseBody
    @PreAuthorize("hasRole('MEETING_READ')")
    public String events(Authentication authentication, @PathVariable Long id,
       HttpServletRequest request) {
       return new CustomEventsManager(request, eventService, homeownerAssociationService,
       id, authentication).run();
    }
}
```

6.8.2 Model uprawnień

W związku z przyjętymi wymaganiami, między innymi możliwości obsługi wielu wspólnot jednocześnie, pojawiła się konieczność zapewnienia szczegółowej kontroli dostępu do aplikacji. Okazało się jednak, że niekiedy poziom, na którym sprawdzane są uprawnienia, nie dotyczy całej wspólnoty, lecz konkretnego mieszkania - tak jest dla właściciela mieszkania lub lokatora. Dodatkowo, różni aktorzy mogą mieć różne zestawy uprawnień. W związku z tym przyjęliśmy następujące założenia do generycznego rozwiązania:

a. musi istnieć możliwość definiowania roli na różnych poziomach, minimum wspólnota oraz mieszkanie
b. musi istnieć możliwość definiowania zestawu uprawnień dla każdej roli
c. użytkownik nie może mieć większych uprawnień, niż wynika to z sumy ról przez niego posiadanych, które odnoszą się do danego obiektu
d. uprawnienia mogą się w dowolny sposób zmieniać w czasie, tzn. przy założeniu spełnienia punktu c musi istnieć możliwość osiągnięcia dowolnego stanu uprawnień użytkownika dostępnego dla danej roli
e. musi istnieć możliwość ograniczania dostępu do zasobów aplikacji na podstawie uprawnień
f. musi istnieć możliwość ograniczania wyników zapytania do obiektów dostępnych do użytkownika
g. nie każdy typ obiektu biznesowy musi mieć ograniczany dostęp
h. musi istnieć możliwość przekazywania dodatkowych atrybutów dla każdej instancji przypisania roli.

Analiza wykazała, że potrzebne jest 6 ról - administrator systemu, zarządca, księgowa, członek zarządu (poziom wspólnoty) oraz lokator i właściciel (poziom mieszkania).W celu realizacji, zostały wprowadzone 3 encje - Role (reprezentująca rolę, słownikowa), Permission (reprezentująca uprawnienie, słownikowa) oraz Person (reprezentująca osobę).Powiązania są następujące:

- Person → Permission - 0..1:0..n
- Role → Permission - 1:n (uwaga - jest to zależność słownikowa uprawnień dostępnych dla roli)

Brakujące powiązanie to połączenie (przypisanie) użytkownika do roli, przy ograniczeniu do konkretnej wspólnoty lub mieszkania.

Z powyższych punktów wynikają następujące wytyczne:

- Jedyna w miarę wydajna implementacja wymagań e i f musi być w kontekście aplikacji zrealizowana przy pomocy Criteria API lub widoków. Wykorzystana implementacja korzysta z Criteria API ze względu na większe reużycie kodu (widoki wymagają napisania szczegółowych zapytań do każdego obiektu, dla którego dostęp jest kontrolowany)
- W przypadku kontrolerów i widoków szczegółowa kontrola dostępu do obiektu wymaga wskazania tego obiektu - wykorzystano tutaj interfejs PermissionResolver z modułu Spring Security.
- Klucz główny "roli" musi uwzględniać datę (na podstawie wymagania d).
- Ze względu na fakt, że różne obiekty biznesowe potencjalnie mogą mieć różne klucze, potrzebne jest rozwiązanie, które będzie parametryzowane typami: encji, na poziomie której sprawdzany jest dostęp, jej klucza, encji, do której dostęp jest sprawdzany oraz jej kluczem. Parametryzacja 4 typami łącznie jest niewygodna, więc jest stosowana jedynie na najwyższych poziomach hierarchii klas implementujących model uprawnień, na niższych szczeblach znajdują się specjalizacje dla konkretnych typów. Drugą konsekwencją jest fakt, że ponieważ klucze encji, na poziomie której sprawdzany jest dostęp, mogą być różnych typów, mogą im odpowiadać różne tabele, wynika to również częściowo z punktu h.

W poniższym opisie będzie stosowane nazewnictwo:

- encja upoważniająca - encja, na poziomie której sprawdzana jest rola i/lub uprawnienie.
- encja udostępniana - encja, do której dostęp jest sprawdzany

Przyjęte rozwiązanie korzysta z następujących klas modelu:

- TimePersonCompositeKey - klucz złożony (@Embeddable) zbudowany z id osoby, id encji (typ parametryzowany) i czasu początku obowiązywania. W aplikacji ma 2 specjalizacje, według konkretnych encji upoważniających - PersonApartmentId (dla mieszkania) i PersonAssociationId (dla wspólnoty)
- EntityAssignableRole - encja, której kluczem jest TimePersonCompositeKey. Reprezentuje pojedyncze przypisanie konkretnej osoby do danej roli od danej daty. Ma dwie bezpośrednie abstrakcyjne podklasy - ApartmentAssignableRole i AssociationAssignableRole. ApartmentAssignableRole ma 2 konkretne podklasy - Homeowner (przypisanie właściciela do mieszkania, dodatkowy atrybut - udział) oraz Tenant (przypisanie lokatora do mieszkania). AssociationAssignableRole ma 4 konkretne podklasy - Sysadmin (administrator systemu dla wspólnoty), BoardMember (członek zarządu wspólnoty), EstateAdministrator (zarządca nieruchomości dla wspólnoty) oraz Accountant (księgowa dla wspólnoty).

Powyższy opis definiuje model powiązania roli z uprawnieniami. Zgodnie z poprzednimi uwagami, aplikacja musi pilnować, aby uprawnienia wyznaczone z relacji Person ↔ Permission nie zawierały nigdy elementów, które nie wynikają z relacji Person ↔ Role (na podstawie EntityAssignableRole w danym momencie) oraz Role ↔ Permission.

Do obsługi tego modelu zostały stworzone odpowiednie klasy DAO, usług i kontrolerów, których hierarchia odzwierciedla hierarchię klasy EntityAssignableRole.

Do kompletnej realizacji potrzebne było również powiązanie kryteriów wyszukiwania dla poszczególnych encji udostępnianych z zestawem uprawnień, tzn. wskazanie, w jaki sposób z danej encji udostępnianej "dotrzeć" do danej encji upoważniającej. Już samo sformułowanie tego pytania wskazuje, że każde DAO dla danej encji udostępnianej musi inaczej obsłużyć przypadek powiązania

ze wspólnotą, a inaczej z mieszkaniem. Takie 2 powiązania są zdefiniowane w DAO dla każdej encji udostępnianej, ale tylko w formie implementacji. Na poziomie API dostępna jest jedna wspólna metoda - getPermissionRestriction, która zwraca obiekt Criterion, który musi być dodany do kryteriów zapytania, aby zawęzić listę dostępnych obiektów do tych, do których osoba, która korzysta z aplikacji, ma dane uprawnienie - jest to realizacja wymagania f. Zastosowanie tego kryterium do encji udostępnianej pozwala automatycznie zrealizować również wymaganie g - weryfikacji dostępu do konkretnego obiektu z danym uprawnieniem. Kryterium to zawiera podzapytanie, które jest unią zapytań na poszczególnych tabelach odpowiadających konkretnym podklasom EntityAssignableRole, skąd wyciągane są tylko aktywne rekordy.

Obiekt Spring Security zarządzający weryfikacją uprawnień w kontrolerach i widokach (obiekt permissionCheckFacade deleguje weryfikację uprawnień do właściwego DAO, wykorzystany został wzorzec projektowy odwiedzającego) przedstawiony na Listing 10.

Listing 10. Zarządzanie uprawnieniami. ***Źródło: Opracowanie własne.***

```
@Service("permissionEvaluator")
public class Application PermissionEvaluator implements PermissionEvaluator {

    @Autowired
    private PermissionCheckFacade permissionCheckFacade;

    @Override
    public boolean hasPermission(Authentication authentication, Object targetDomainObject,
            Object permission) {
        UserPermission userPermission = getPermission(permission);
        if (targetDomainObjectinstanceof  {
            AbstractEntity
        })
        {
              PermissionContext permissionContextUserPermission
              .createContext(authentication,
                       (AbstractEntity)targetDomainObject);
              return permissionCheckFacade.visit(permissionContext);
        }
        else if(targetDomainObject != null) {
        throw new IllegalArgumentException("Obiekt typu " +
        targetDomainObject.getClass().getSimpleName() + " nie może być zweryfikowany.");
        }
// nie ma dostępu do nulla.
        return false;
    }

    @Override
    public boolean hasPermission(Authentication authentication, Serializable targetId,
            String targetType, Object permission) {
        UserPermission userPermission = getPermission(permission);
        if (targetId != null) {
            PermissionContext permissionContext
                    =userPermission.createContext(authentication, targetId);
            return permissionCheckFacade.visit(permissionContext);
        } else { // niemadostępu do nulla.
            return false;
        }
    }

    private UserPermission getPermission(Object permission) {
        return UserPermission.valueOf(UserPermission.class,
```

```
                String.valueOf(permission).toUpperCase());
    }
// Interfejs DAO kontrolującego dostęp do zarządzanej encji:
    public interface SecuredEntityDAO {

        default Set<UserPermission> getAllPermissions() {
            return EnumSet.of(getCreatePermission(),
                    getDeletePermission(),
                    getReadPermission(),
                    getUpdatePermission());
        }

        UserPermission getReadPermission();
        UserPermission getDeletePermission();
        UserPermission getCreatePermission();
        UserPermission getUpdatePermission();

        Criterion getPermissionRestriction
                    (Authentication authentication, UserPermission... permissions);
        Set<Serializable> getPermittedObjects(PermissionContext permissionContext);
        default boolean validatePermissionContext(PermissionContext permissionContext) {
            if (!getAllPermissions().contains(permissionContext.getPermission())) {
                throw new UnsupportedOperationException("Nieobsługiwane uprawnienie "
                          + permissionContext.getPermission()
                          + " w dao: " + getClass().getSimpleName());
            } else {
                return true;
            }
        }
    }
}
```

Metoda zwracająca ograniczenia na obiekty dostępne dla użytkownika w zapytaniu (klasa AbstractSecuredDAO, dziedzicząca po AbstractEntityDAO, implementująca SecuredEntityDAO) została przedstawiona na Listing 11

Listing 11. Metoda zawierająca ograniczenia. ***Źródło: Opracowanie własne.***

```
public Criterion getPermissionRestriction(String userName, UserPermission... permissions){
       String[] permissionNames = new String[permissions.length];
        for (int i = 0; i<permissions.length; i++) {
          permissionNames[i] = permissions[i].name();
         }
       List<Criterion>components = new ArrayList<>();
        for (UserRoleAccessLevel accessLevel: UserRoleAccessLevel.values()) {
           components.add(Subqueries.propertyIn("id", getAccessLevelCondition
                     (accessLevel, PERMITTING_ENTITY_ALIAS)
           .add(getPermissionSubqueryForAccessLevel(accessLevel,
                     PERMITTING_ENTITY_ALIAS + ".id", userName, permissionNames))
           .setProjection(getAccessedEntityIdProjectionForAccessLevel
                     (accessLevel, PERMITTING_ENTITY_ALIAS + ".id"))));
         }
    return Restrictions.disjunction(components.toArray(new Criterion[components.size()]));
}
```

6.8.3 Filtrowanie

W ramach ułatwienia pracy z aplikacją w każdym widoku przedstawiającym listy danych, zostało zaimplementowane pole umożliwiające filtrowanie. Działanie mechanizmów zrealizowane jest

po stronie klienta za pomocą JavaScriptu. Taki sposób implementacji zapewnia niskie zużycie zasobów przez mechanizm.

Mechanizm składa się z dwóch części pola o nazwie „Filtr" które przyjmuje wartości do filtrowania wpisane przez użytkownika, następnie wpisana wartość przekazywana jest do JavaScriptu, który odpowiada za przefiltrowanie danych.

Pole filtrujące zostało zaimplementowane w następujący sposób: typem pola jest search.: typem pola jest search. Klasa przypisana do kontrolki to „light-table-filter", za pomocą klasy kontrolki zrealizowane jest połączenie kontrolka-skrypt. Następnym polem kontrolki jest „data-table" którego wartość to „order-table", pole to określa do jakiej tabeli przypisana jest kontrolka filtrująca tj. jaki zakres danych ma zostać przefiltrowany. Ostatnim polem jest „placeholder" z wartością „Filtr", które określa domyślną wiadomość pokazującą się w kontrolce.

Listing 12. Implementacja pola filtrującego. ***Źródło: Opracowanie własne.***

```
<input type="search"class="light-table-filter"data-table="order-table" placeholder="Filtr"
```

Skrypt filtrujący jest przypisany do kontrolki o klasie „light-table-filter", skrypt reaguje na wpisanie treści do kontrolki. Następnie skrypt odnajduje element o klasie „data-table", którym jest tabela mająca zostać przefiltrowania. Po wprowadzeniu tekstu do kontrolki filtrujące wiersze sprawdzanej tabeli zostają sprawdzone pod kątem zawierania wpisanej treści. Wiersze, które nie zawierają wyszukiwanej treści są ukrywane.

Listing 13. Skrypt filtrujący. ***Źródło: Opracowanie własne.***

```
(function (document) {
    'usestrict';
    var LightTableFilter = (function (Arr) {
        var _input;
        function _onInputEvent(e) {
            _input = e.target;
                    var tables = document.getElementsByClassName
                                (_input.getAttribute('datatable');
                            Arr.forEach.call(tables, function (table) {
                                Arr.forEach.call(table.tBodies, function (tbody) {
                                    Arr.forEach.call(tbody.rows, _filter);
                                });
                            });
                }
                function _filter(row) {
                    var text = row.textContent.toLowerCase(),
                            val = _input.value.toLowerCase();
                    row.style.display = text.indexOf(val) === -1 ? 'none' : 'table-row';
                }
                return {
                    init: function () {
                       var inputs = document.getElementsByClassName('light-table-filter');
                       Arr.forEach.call(inputs, function (input) {
                           input.oninput = _onInputEvent;
                        });
                    }
                };
            })(Array.prototype);
    document.addEventListener('readystatechange', function () {
        if (document.readyState === 'complete') {
            LightTableFilter.init();
        }
```

```
    });
    });
)(document);
```

6.8.4 Statystyki

Widok statystyk zgodnie z intuicyjnym rozumieniem przedstawia zestaw kilku najważniejszych agregatów danych opisujących sytuację (m.in. finansową) w ramach danej wspólnoty mieszkaniowej. Na ekranie dane zostały podzielone na trzy grupy: informacje o płatnościach, licznikach oraz o wspólnocie.

Statystyki są standardowym dokumentem HTML wzbogaconym o dyrektywy FreeMarker Java Template Engine odpowiadające przede wszystkim za odpowiednie powiązanie danych statystycznych (znajdujących się w klasie modelu) do elementów DOM strony. Sam model jest prostą klasą Javy, zawierającym atrybuty opisujące konkretne agregaty statystyk (Listing 14).

Listing 14. Model z klasa i atrybutami. ***Źródło: Opracowanie własne.***

```
public class StatisticsViewModel {
    private double paymentsWeekTotal;
    private double paymentsWeekAvg;
    private double paymentsMonthTotal;
    private double paymentsMonthAvg;
    private double meterStatesWeekTotal;
    private double meterStatesWeekAvg;
    private double meterStatesMonthTotal;
    private double meterStatesMonthAvg;
    private long buildingsCount;
    private long tenantsCount;
    private long flatsCount;
}
```

Model wypełniony danymi do widoku przekazywany jest za pomocą metody dedykowanego kontrolera, czyli odpowiedniej klasie napisanej w oparciu o Spring MVC. Metoda ta jest z kolei powiązana z adresem URL strony, na której znajdują się statystyki, dzięki czemu w momencie otwarcia danego adresu w przeglądarce kontroler wie, że właśnie w tym momencie powinien uruchomić proces obliczenia statystyk, a wyniki przekazać odpowiedniemu widokowi.

Listing 15. Obliczanie statystyk i przekazywanie widoku. ***Źródło: Opracowanie własne.***

```
@Controller
@RequestMapping("/secure/statistics")
public class StatisticsController {

    @RequestMapping(method = RequestMethod.GET, value = "/{id}")
    @PreAuthorize("hasRole('STATISTICS_READ') &&hasPermission(#id,'','STATISTICS_READ')")
    public String get(@ModelAttribute("model") ModelMapmodel, @PathVariable Long id,
            RedirectAttributesredirectAttributes, Authentication authentication){
        HomeownerAssociation ha = homeownerAssociationService.getInstance(id);
        List < Long > homeownerAssociationIdsForUser
                =homeownerAssociationService.getUserAssociationIds
                (authentication, UserPermission.STATISTICS_READ);
        model.addAttribute("homeownerAssociations", homeownerAssociationService.list
                    (Restrictions.in("id", homeownerAssociationIdsForUser)));
        model.addAttribute("selectedAssociationId", id);
        model.addAttribute("statistics", statsService.getStatistics(id));
        model.addAttribute("Gaz", chartDataService.getMeterChart(id, "Gaz"));
        model.addAttribute("Prąd", chartDataService.getMeterChart(id, "Prąd"));
        model.addAttribute("Woda_Z", chartDataService.getMeterChart(id, "Woda zimna"));
```

```
            model.addAttribute("Woda_C", chartDataService.getMeterChart(id,"Woda Ciepła"))
            return"statistics/details";
        }
}
```

Kontroler przekazuje zadanie wygenerowanie statystyk do odpowiedniego serwisu (w tym wypadku **StatisticsService**), który znajdując się w warstwie usług aplikacji posiada dostęp do warstwy dostępu do danych (DAO) i to właśnie z niej wywołuje poszczególne metody odpowiedzialne za obliczenie odpowiednich wartości statystycznych. Po otrzymaniu wyników wypełnia nowoutworzony obiekt klasy modelu i przekazuje go wyżej, do kontrolera.

Listing 16. Serwis StatisticsServce. *Źródło: Opracowanie własne.*

```
@Component
@Transactional
public class StatisticsService {
private final StatisticsDAO statisticsDAO;
        public StatisticsViewModelgetStatistics(Long homeownerAssociationId){
             StatisticsViewModelstats = newStatisticsViewModel();
            statisticsDAO.updatePaymentStatistics(stats, homeownerAssociationId);
            statisticsDAO.updateMeterStateStatistics(stats, homeownerAssociationId);
            statisticsDAO.updateBuildingStatistics(stats, homeownerAssociationId);
            statisticsDAO.updateApartmentStatistics(stats, homeownerAssociationId);
            return stats;
        }
}
```

Następnie, w klasie warstwy DAO, dla każdej z jej metod wywoływane jest odpowiednie zapytanie NHibernate-a, który ostatecznie tłumaczy je na odpowiednie zapytanie SQL obliczające zagregowane wartości statystyk. Przykładowa metoda DAO przedstawiona została na Listing 17.

Listing 17. Przykładowa metoda DAO. *Źródło: Opracowanie własne.*

```
public void updatePaymentStatistics(StatisticsViewModelstats,Long homeownerAssociationId){
Object[] result = (Object[]) hibernateTemplate.findByNamedQueryAndNamedParam(
        "getPaymentStatistics",
    new String[]{"assocId", "minCreateDate"},new Object[]{homeownerAssociationId,
                DateHelper.toSqlDate(LocalDate.now() .minusMonths(1))}).iterator().next();
        BigDecimalsum = (BigDecimal) result[0];
        Double avg = (Double) result[1];
        stats.setPaymentsMonthAvg(nvl(avg));
        stats.setPaymentsMonthTotal(nvl(sum).doubleValue());
        result = (Object[]) hibernateTemplate.findByNamedQueryAndNamedParam
                ("getPaymentStatistics", new String[]{"assocId", "minCreateDate"},new
                Object[]{homeownerAssociationId,DateHelper.toSqlDate
                (LocalDate.now().minusDays(7))}).iterator().next();
        sum = (BigDecimal) result[0];
        avg = (Double) result[1];
        stats.setPaymentsWeekAvg(nvl(avg));
        stats.setPaymentsWeekTotal(nvl(sum).doubleValue());
}
```

6.8.5 Widok

Widok „Moje mieszkanie” przedstawia najważniejsze informacje dla właściciela mieszkania. Znajdują się w nim podstawowe informacje o mieszkaniu, szczegółowe informacje o licznikach, a także informacje o dokonywanych płatnościach na rzecz mieszkania.

Za dostarczenie niezbędnych informacji do widoku odpowiada kontroler **MyApartment** i jego metoda **addForm**. Metoda pobiera informacje o zalogowanym użytkowniku, następnie na podstawie tych informacji, pobiera przypisane do użytkowania mieszkania i przekazuje dane do widoku.

Listing 18. Kontroler MyApartment i jego metoda. ***Źródło: Opracowanie własne.***

```
@Controller
public class MyApartmentController {

     private final ApartmentServiceapartmentService;
     private final PersonServicepersonService;
     private final AccessRightsServiceaccessRightsService;

     @Autowired
     public MyApartmentController(PersonServicepersonService,
             AccessRightsServiceaccessRightsService,
             ApartmentServiceapartmentService) {
        this.personService = personService;
        this.accessRightsService = accessRightsService;
        this.apartmentService = apartmentService;
     }

     @RequestMapping(value = "/myApartment", method = RequestMethod.GET)
     public String addForm(@ModelAttribute("model") ModelMap model) {
        Authenticationauthentication = SecurityContextHolder.getContext()
                .getAuthentication();
        String username = authentication.getName()
        Person person = personService.loadUserByUsername(username);
        model.addAttribute("person", person);
        List < Apartment > owned = accessRightsService.getApartmentsForHomeowner(person);
        owned = apartmentService.listWithDeepDependencies(owned.stream()
                .map(DefaultEntity::getId).collect(Collectors. < Long > toList()));
        model.addAttribute("owned", owned);
        model.addAttribute("osize", owned.size());
        return "myApartment";
     }
}
```

Dzięki metodzie **addNumber** system zapisuje informacje na temat liczby lokatorów. Metoda ta pobiera informacje o zalogowanym użytkowniku , następnie na podstawie tych danych, przypisuje wpisaną przez użytkownika wartość i przekazuje ją do systemu.

Listing 19. Metoda addNumber. ***Źródło: Opracowanie własne.***

```
@RequestMapping(value = "/myApartment/{appid}/{number}", method = RequestMethod.GET)
public String addNumber(@PathVariableintnumber, @PathVariableLongappid,
        @ModelAttribute("model") ModelMapmodel) {
    Authenticationauthentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    Person person = personService.loadUserByUsername(username);
    model.addAttribute("person", person);
    Apartmentapp = apartmentService.getInstance(appid);
    if (!accessRightsService.isHomeownerFor(person, app)) {
        return"";
    }
    apartmentService.updateApartment(app, number);
    return"redirect:/web/myApartment/";
```

```
}
```

Do deklaracji wykorzystane zostało okno dialogowe, odpowiada za to klasa „modal-dialog” (Listing 20) dzięki której okno wyświetlane jest na ekranie na tej samej podstronie.

Listing 20. Modal dialog. ***Źródło: Opracowanie własne.***

```
<div class = "modalfade" id = "populationWindow-${apartment.id}" tabindex="1"
     role="dialog" aria - hidden = "true">
     <div class = "modal-dialog">
        <form onsubmit = "location.href='/szw/web/myApartment/${apartment.id}/' +"
           +" document.getElementById('population-${apartment.id}').value; return false;">
              <div class = "modal-content">
                <div class = "modal-header">
                    <button type = "button" class = "close"
                            data-dismiss= "modal"
                            aria label="Close">
                        <span aria - hidden = "true" > & times; </span>
                    </button>
                    <h4 class = "modal-title"> Deklaracja liczby mieszkańców.
                        <b><font color ="red">UWAGA!</font></b>
                        Zapisanie deklaracji jest jednocześnie oświadczeniem, że
                        wpisane dane są zgodne ze stanem rzeczywistym. Jest również
                        potwierdzeniem świadomości, że składanie fałszywych oświadczeń
                        grozi odpowiedzialnością karną. Składający deklarację
                        zobowiązuje się do niezwłocznego informowania zarządu
                        wspólnoty o każdej zmianie liczby mieszkańców zamieszkujących
                        w lokalu w terminie 7 dni od zaistnienia tej zmiany.
                    </h4>
                    <div class = "modal-body">
                        <div class = "input-group">
                            <label for ="population-${apartment.id}">
                                Liczba mieszkańców:
                            </label>
                            <input type ="text" class= "form-control"
                                   id="population${apartment.id}"
                                   value = "${apartment.population}"
                                   pattern = "^\d+$">
                        </div>
                    </div>
                    <div class = "modal-footer">
                        <button type= "button" class="btnbtn-default"
                                Data - dismiss="modal">Anuluj </button>
                    </div>
                </div>
                <input type = "submit" class = "btnbtn-primary"
                       value = "Zapisz deklarację">
            </div>
    </div>
</div>
```

Listing 21 odpowiada za wyświetlenie stanów liczników należących do mieszkania. Dzięki zastosowaniu responsywnych tabel stany liczników wyświetlają się poprawnie na wszystkich urządzeniach. System pobiera informacje o stanach liczników. Domyślnie wyświetlane są informacje o 5 ostatnich operacjach. Użytkownik może przejść do przeglądania wszystkich liczników.

Listing 21. Wyświetlanie stanów licznika. *Źródło: Opracowanie własne.*

```
<h4>Liczniki:</h4>
<div id="meters" >
    <table class="tabletable-stripedtable-hovertable-responsive">
        <thead>
            <tr>
                <th></th>
                <th data-field="date">Typ</th>
                <th data-field="content">Numer Seryjny</th>
                <th></th>
            </tr>
        </thead>

        <tbody>
            <#list apartment.meter?sort_by("serialNumber") as meter>
            <tr onClick="
                    showhide('${meter.serialNumber}');
                    showhide('down1${meter.serialNumber}');
                    showhide('up1${meter.serialNumber}');
                    showhide('down2${meter.serialNumber}');
                    showhide('up2${meter.serialNumber}');">

                <td>
                    <span id="down1${meter.serialNumber}"
                          class="glyphiconglyphicon-chevron-down"
                        aria-hidden="true">
                    </span>
                    <span id="up1${meter.serialNumber}" style="display:none;"
                          class="glyphiconglyphicon-chevron-up"
                        aria-hidden="true">
                    </span>
                </td>

                <td>${meter.type!}</td>
                <td>${meter.serialNumber!}</td>
                <td style="text-align: right;">
                    <span id="down2${meter.serialNumber}"
                          class="glyphiconglyphicon-chevron-down"
                        aria-hidden="true">
                    </span>
                    <span  id="up2${meter.serialNumber}" style="display:none;"
                           class="glyphiconglyphicon-chevron-up"
                        aria-hidden="true">
                    </span>
                </td>
            </tr>

            <tr id="${meter.serialNumber}" style="display:none;" >
                <td colspan="4">
                    <table class="tabletable-stripedtable-hovertable-responsive">
                        <thead>
                            <tr>
                                <th data-field="meter">Licznik</th>
                                <th data-field="dateFrom">Data od</th>
                                <th data-field="dateTo">Data do</th>
```

```
                                <th data-field="value">Stan</th>
                                <th data-field="difference">Zużycie</th>
                                <th data-field="charge">Opłata</th>
                                <th data-field="pricerate">Cena za jednostke</th>
                                <th data-field="accepted">Akceptacja księgowości</th>
                            </tr>
                        </thead>
                    </table>
                </td>
            </tr>
        </tbody>
    </table>
</div>
```

Listing 22 odpowiada za wyświetlenie płatności należących do mieszkania. System pobiera informacje o płatnościach. Wyświetlane są informacje o 5 ostatnich operacjach. Użytkownik może przejść do przeglądania wszystkich płatności.

Listing 22. Wyświetlanie płatności należących do mieszkania. ***Źródło: Opracowanie własne.***

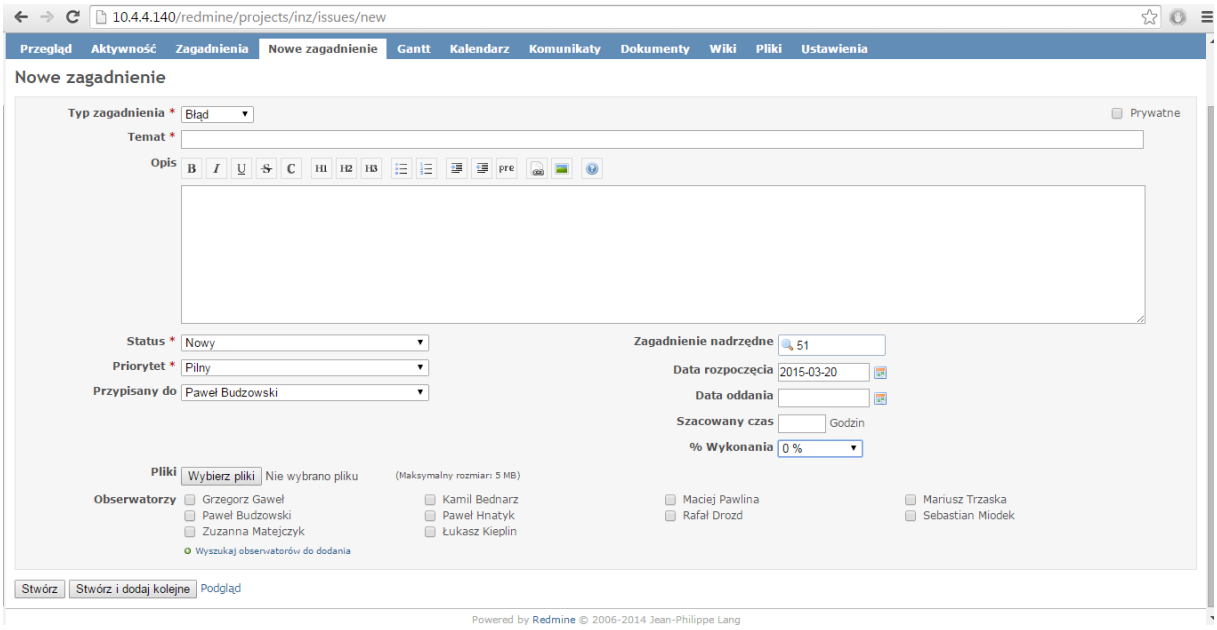
```
<h4>Ostatnie 5 wpłat:</h4>
<button type="button" class="btnbtn-default"onclick="showhidem('hidpayment','nonthing');"
    >Pokaż wszystkie</button>
<table class="tabletable-stripedtable-hovertable-responsive order-table">
    <thead>
        <tr>
            <th data-field="payer">Płatnik</th>
            <th data-field="created">Data Płatności:</th>
            <th data-field="content">Kwota</th>
            <th data-field="name">Tytuł płatności</th>
            <th data-field="attachment">Komentarz</th>
            <th colspan="3"></th>
        </tr>
    </thead>
    <tbody>
        <#list apartment.payment?sort_by("created")?reverse as payment>
        <#ifpayment_index = 5><#break></#if>
        <tr title="Szczegóły" onClick="moreless('${payment.id}');">
            <td>${payment.payer}</td>
            <td>${payment.created}</td>
            <td align="right" class="money">${payment.amount}zł</td>
            <td name="${payment.id}" style="white-space: nowrap; max-width: 12em;
                overflow:  hidden; text-overflow: ellipsis">
                ${payment.paymentTitle}</td>
            <td name="${payment.id}"style="white-space: nowrap;max-width: 12em;
                overflow: hidden; text-overflow: ellipsis;">
                ${payment.comments}</td>
        </tr>
        </#list>
        <#list apartment.payment?sort_by("created")?reverse as payment>
        <#ifpayment_indexgt 5>
        <tr name="hidpayment" style= "display:none;" title="Szczegóły"
            onClick="moreless('${payment.id}');">
            <td>${payment.payer}</td>
            <td>${payment.created}</td>
            <td align="right" class ="money">${payment.amount}zł</td>
            <td name="${payment.id}"style="white-space: nowrap;
                max-width: 12em; overflow: hidden; text-overflow: ellipsis">
```

```
                ${payment.comments}</td>
            <td name="${payment.id}"style="white-space: nowrap;
                max-width: 12em overflow: hidden; text-overflow: ellipsis">
                ${payment.paymentTitle}</td>
        </tr>
        </#if>
        </#list>
    </tbody>
</table>
```

7. Testy funkcjonalne

Głównym zadaniem testów jest zapewnienie jakości wytwarzanego oprogramowania oraz sprawdzenie, czy wytwarzane oprogramowanie jest zgodne z celami i założeniami projektu. Testowanie oprogramowania zostało rozpoczęte zaraz po zdefiniowaniu wymagań oraz po zaimplementowaniu pierwszych funkcjonalności. Pozwoliło to uniknąć błędów, które w późniejszym czasie byłyby kosztowniejsze w naprawie. Przebieg testów był realizowany za pomocą scenariuszy testowych, które krok po kroku opisywały czynności, które należało wykonać, aby test został zakończony. O wyniku testów decydowały kryteria akceptacji.

W sytuacji gdy stwierdzono błędy, osoba odpowiedzialna za realizowanie poszczególnych scenariuszy testowych, dokonywała rejestracji błędu w systemie zarządzania projektami Redmine. Realizacja takiego zgłoszenia polegała na nadaniu tematu, opisu, priorytetu naprawy oraz przypisaniu błędu do osoby odpowiedzialnej za naprawę (Rysunek 94).



Rysunek 94. Dodawanie błędu do Redmine. ***Źródło: Opracowanie własne.***

Błędy zostały podzielone na cztery klasy. Każda klasa błędu została opisana i nadano jej priorytet naprawy (Tabela 9).

Tabela 9. Kwalifikacja błędów. *Źródło: Opracowanie własne.*

Klasa błędu	Opis błędu
1	Najwyższa klasa błędu, który uniemożliwia dalszą pracę, jest krytyczny ze względu na funkcjonowanie aplikacji. Priorytet: natychmiastowy.
2	Jednoznaczny błąd, który nie powoduje blokowania naszej pracy, a zarazem nie jest krytyczny ze względu na funkcjonowanie aplikacji. Priorytet: pilny.
3	Błąd, który nie wpływa na działanie aplikacji, ale nie jest zgodny z wymaganiami. Priorytet: wysoki.
4	Błąd najniższej klasy, który nie ma wpływu na funkcjonowanie aplikacji, są to np. literówki. Priorytet: normalny.

7.1. Faza 1 – CRUD

Celem pierwszej fazy testów było przetestowanie tworzenia, czytania, aktualizowania i kasowania wszystkich obiektów. Jej nazwa pochodzi od pierwszych liter w języku ang. *Create*, *Read*, *Update*, *Delete*. Poniżej przedstawiono kilka najciekawszych scenariuszy testowych (Tabela 10-11). Załącznik 1 zawiera wszystkie tabele testowe fazy CRUD.

Tabela 10. CRUD obiektu ogłoszenia. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Ogłoszenia”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z lewego menu kliknij "Ogłoszenia" a następnie wybierz "Dodaj ogłoszenie"	
2	Wybierz "Dodaj ogłoszenie" bez wpisywania danych	
3	Wypełnij dostępne pola	1.Kategoria ogłoszenia nie wskazuje wartości
		2. Brak pola „Temat”
4	Wybierz "Dodaj ogłoszenie"	1. Brak „Anuluj”

5	Spróbuj ponownie dodać takie samo ogłoszenie	1. Możliwość dodania kilkukrotnie tego samego ogłoszenia z tymi samymi danymi
6	Wybierz z menu "Ogłoszenia" następnie "Wyświetl ogłoszenia"	1. Dodane ogłoszenia są wyświetlane w odwrotnej kolejności
7	W panelu Ogłoszeń wybierz "Edytuj"	1.Brak opcji „Edytuj”
8	W panelu Ogłoszeń wybierz "Usuń"	1. Brak opcji „Usuń”
9	W panelu Ogłoszeń wybierz "Wyszukaj"	1. Brak opcji „Wyszukaj”
Oczekiwany rezultat	1.Dodanie ogłoszenia 2. Wyświetlenie ogłoszeń 3. Edycja ogłoszeń 4. Usuń ogłoszenie	
Uzyskany rezultat	Niezgodny z oczekiwaniem	
Klasa błędu	1	

Tabela 11. CRUD obiektu operacje finansowe. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu "Operacje finansowe"	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z lewego menu kliknij "Operacje finansowe" a następnie wybierz "Dodaj Operacje"	
2	Wybierz "Dodaj" bez wpisywania danych	
3	Wybierz "Anuluj"	Brak powrotu
4	Wybierz "Dodaj"	
5	Wypełnij dostępne pola i "Dodaj operacje finansowa"	
6	Ponów krok 5 z wpisując takie same dane	Można dodać 2 x taka sama operacje
7	Kliknij "Wydatki Inwestycyjne" a następnie wybierz "Przeglądaj Operacje"	

8	Wybierz "Szczegóły" operacji	Internal Server Error
9	Wybierz "Edytuj" operacje	Brak operacji "Zapisz" i "Anuluj"
10	Wybierz "Usuń" operacje	Internal Server Error
Oczekiwany rezultat	1. Dodanie Wydatku 2.Edycja Wydatku 3. Usuniecie Wydatku 4. Przeglądanie Wydatków	
Uzyskany rezultat	negatywny	
Klasa błędu	1	
Wynik testu	Wymaga pilnej naprawy	

Tabela 12. CRUD obiektu wspólnota. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu "Wspólnota"	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z lewego menu kliknij "Wspólnota" a następnie wybierz "Dodaj wspólnotę"	
2	Wybierz "Dodaj" bez wpisywania danych	
3	Wypełnij dostępne pola	1.Przy wypełnianiu kwestionowano sens pola "Dane adresowe"
4	Wybierz "Dodaj"	
5	Spróbuj ponownie dodać taka samą wspólnotę	1.Wyskakuje błąd "Internal Server Error" mało czytelny dla użytkownika
6	Wybierz z menu "Wspólnota" następnie "Wyświetl wspólnoty"	
7	Wyświetl dodana wspólnotę	
8	W panelu zarządzania wspólnotami wybierz "Szczegóły"	1. Brak powrotu z panelu "Szczegóły"
9	W panelu zarządzania wspólnotami wybierz "Edytuj"	1. Brak "Anuluj" w Edycji

10	Wybierz "Usuń"	1. W przypadku usunięcia zdiagnozowano błąd niestety nie udało się go powtórzyć
Oczekiwany rezultat	1.Dodanie wspólnoty 2. Wyświetlenie wspólnoty 3. Edycja wspólnoty 4. Usuń wspólnotę	1. Wyszukaj
Uzyskany rezultat	Negatywny	
Klasa błędu	1	
Wynik testu	Wymaga pilnej naprawy	

7.2. Faza 2 – Weryfikacja wprowadzanych danych

Zadaniem drugiej fazy było przetestowanie, czy wprowadzone dane są trwałe, zapisują i wyświetlają się poprawnie oraz posiadają odpowiedni format. Tabela 13 do Tabela 15 zawierają testowane elementy, zaś pozostała cześć została przeniesiona do Załącznik 2.

Tabela 13. Obiekt wspólnota. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych – obiekt Wspólnota	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Panel administracyjny", następnie wybierz "Wspólnota", a na końcu "+" oznaczający dodanie nowej wspólnoty	
2	Wypełnić następujące pola: • Nazwa wspólnoty • Dane adresowe • Nazwa ulicy • Numer budynku • Numer lokalu • Kod pocztowy • Miejscowość	
3	Wybierz Dodaj	
Oczekiwany rezultat	Zarejestrowano nowy obiekt	

	Wspólnota w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Wspólnota w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 14. Obiekt ogłoszenia. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Ogłoszenia	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Ogłoszenia", a na końcu "+" oznaczający dodanie nowego ogłoszenia.	
2	Wypełnić następujące pola: • Tytuł • Treść • Kategoria • Kategoria • Priorytet	
3	Wybierz "Dodaj"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Ogłoszenia w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Ogłoszenia w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 15. Obiekt uchwały. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Uchwały	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Uchwały", a na końcu "+" oznaczający dodanie nowej uchwały/	

2	Wypełnić następujące pola: • Nazwa • Data uchwalenia • Treść • Załącznik • Uchwalono na spotkaniu	
3	Wybierz "Zapisz"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Uchwały w bazie danych.	
Uzyskany rezultat	Zarejestrowano nowy obiekt Uchwały w bazie danych.	
Klasa błędu	0	
Wynik testu	Pozytywny	

7.3. Faza 3 – Interfejs użytkownika

Zadaniem trzeciej fazy było przetestowanie interfejsu użytkownika. Sprawdzono min. dodawanie wspólnoty oraz przypisanie do niej budynków i mieszkań. Zbadano także poprawność działania wyszukiwania, deklaracji liczby mieszkańców oraz akceptacje projektów inwestycji. Tabela 16 do Tabela 21 zawierają elementy oraz wyniki testowanych elementów.

Tabela 16. Dodanie wspólnoty do systemu. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Dodanie wspólnoty do systemu.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora.	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu głównego wybierz "Panel administracyjny".	
2	Przejdź do zakładki "Wspólnota".	
3	Wybierz „+„ oznaczający dodanie nowej wspólnoty.	
4	Wypełnij następujące pola: Dane wspólnoty: • Nazwa wspólnoty, • stawka czynszu; Adres korespondencyjny: • Ulica, • Numer budynku, • Numer lokalu, • Kod pocztowy, • Województwo, • Miejscowość. Budynki: Zostawić puste	
5	Wybierz przycisk "Zapisz".	

Oczekiwany rezultat	Dodanie wspólnoty do systemu.	
Uzyskany rezultat	Dodanie wspólnoty do systemu.	
Wynik testu	Pozytywny	

Tabela 17. Dodanie budynku i przypisanie do wspólnoty. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Dodanie budynku i przypisanie do wspólnoty.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu głównego wybierz "Panel administracyjny"	
2	Przejdź do zakładki "Budynki"	
3	Wybierz „+„ oznaczający dodanie nowego budynku.	
4	Wypełnij pola: • Ulica, • Numer budynku, • Kod pocztowy, • Województwo, • Miejscowość,	
5	Z sekcji "Przydziel do wspólnoty" wybierz poprzednio dodaną wspólnotę.	
6	Wybierz przycisk "Dodaj budynek"	
7	Z menu głównego wybierz "Panel administracyjny"	
8	Przejdź do zakładki "Wspólnoty" a następnie w szczegółach wspólnoty sprawdź czy dodano budynek.	
Oczekiwany rezultat	Wyświetlenie budynków w liście oraz przypisanie go do wspólnoty	
Uzyskany rezultat	Budynek dodany i przypisany do wspólnoty. Widoczny w szczegółach wspólnoty.	
Wynik testu	Pozytywny	

Tabela 18. Dodanie mieszkania i przypisanie do budynku. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Dodanie mieszkania i przypisanie do budynku.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora.	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**

1	Z menu głównego wybierz "Panel administracyjny".	
2	Przejdź do zakładki "Mieszkania".	
3	Wybierz „+„ oznaczający dodanie nowego mieszkania.	
4	Wypełnij pola: • Numer mieszkania, • Piętro, • Metraż, • Liczba mieszkańców.	
5	Z sekcji "Przydziel do budynku" wybierz budynek dodany w poprzednim scenariuszu.	
6	Wybierz przycisk "Dodaj mieszkanie"	
7	Z menu głównego wybierz "Panel administracyjny".	
8	Przejdź do zakładki "Budynki" a następnie w szczegółach budynków sprawdź czy dodano mieszkanie.	
Oczekiwany rezultat	Wyświetlenie mieszkania w liście oraz przypisanie go do budynku.	
Uzyskany rezultat	Mieszkanie dodane i przypisane do budynku. Widoczne w szczegółach budynku.	
Wynik testu	Pozytywny	

Tabela 19. Wyszukiwanie uchwał. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Wyszukiwanie uchwał po wymaganiach zdefiniowanych przez użytkownika.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora.	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu głównego wybierz "Panel administracyjny".	
2	Przejdź do zakładki "Uchwały".	
3	W oknie po prawej stronie o nazwie "Filter" wpisz szukana frazę.	
Oczekiwany rezultat	Wyświetlone uchwały po zastosowaniu filtru.	
Uzyskany rezultat	Wyświetlone uchwały po zastosowaniu filtru.	

Wynik testu	Pozytywny

Tabela 20. Deklaracja liczby mieszkańców. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Deklaracja nowej liczby mieszkańców.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora.	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu głównego wybierz "Moje mieszkanie".	
2	W panelu mieszkańca kliknij w dowolne miejsce kafla "Zadeklaruj liczbę mieszkańców".	
3	W nowym oknie wpisz liczbę mieszkańców.	
4	Wybierz "Zapisz deklaracje"	
Oczekiwany rezultat	Zmienienie liczby osób przypisanych do mieszkania.	
Uzyskany rezultat	Zmieniono liczbę mieszkańców.	
Wynik testu	Pozytywny	

Tabela 21. Akceptacja projektu i edycja w inwestycjach. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Akceptacja projektu inwestycji, zmiana statusu w inwestycjach, sortowanie po kolumnach.	
Warunki początkowe	Zalogowany użytkownik z prawami administratora.	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu głównego wybierz "Inwestycje".	
2	Przejdź do zakładki "Projekty".	
3	Kliknij ikonę "Zatwierdź" w liście projektów bądź po wejściu w szczegóły.	
4	Z menu głównego wybierz "Inwestycje" .	
5	Przejdź do zakładki "Lista inwestycji".	
6	Wejdź w edycje zaakceptowanego wcześniej projektu.	
7	Wypełnij pola: • Nazwa inwestycji, • Status, • Wartość	

8	Klikając w nazwę tabeli dokonaj sortowania danych.	
Oczekiwany rezultat	Zatwierdzenie projektu i przeniesienie go do inwestycji, zmiana statusu, posortowanie danych.	
Uzyskany rezultat	Status zmieniony na "Zatwierdzony", projekt przeniesiony do inwestycji gdzie zmieniono status i posortowano listę.	
Wynik testu	Pozytywny	

8. Podsumowanie i Wnioski

System zarządzania wspólnotami mieszkaniowymi okazał się projektem, wbrew przewidywaniom, trudniejszy niż na początku nam się to wydawało. Konieczny wydaje się tutaj cytat: „*Projekt to sekwencja niepowtarzalnych, złożonych i związanych ze sobą zadań, mających wspólny cel, przeznaczonych do wykonania w określonym terminie bez przekraczania ustalonego budżetu, zgodnych z założonymi wymaganiami.* [38] ”. A więc wszystkie etapy, począwszy od fazy analizy do dokumentacji, musiały ściśle ze sobą współpracować. Było to duże wyzwanie organizacyjne, które w tym projekcie napotkało szereg problemów. Głównie bolączką naszego przedsięwzięcia okazały się:

- komunikacja wewnątrz zespołowa;
- zróżnicowanie osób, charakterów oraz ich umiejętności;
- nierealizowanie celów skutkujące mało wydajna pracą;
- zniechęcenie uczestników projektu oraz ich podejście do pracy.

Problemy były na bieżąco analizowane i podejmowano kroki, które miały uchronić przedsięwzięcie przed klęską. Osobami odpowiedzialnymi za to był kierownik projektu wraz z kierownikami zespołów. Na bieżąco analizowali ryzyko i starali się dostosować zadania do poziomu kompetencji uczestników. Dbali o dotrzymywanie terminu oraz motywacje w zespołach. Dodatkowym problemem był fakt, iż osobowość ludzka ma charakter dynamiczny, a więc niejednokrotnie trzeba było interweniować w sytuacjach, których próżno szukać w podręcznikowych poradnikach.

Ocenia się, że 7-9 miesięcy w informatyce przynosi zmiany, które w innych dziedzinach zajmują około 5 do 7 lat. Połączenie tego faktu z koniecznością zarządzania zasobem ludzkim niewątpliwie należało do jednego z najtrudniejszych zadań. Słuszne wydają się tu być słowa Henry'ego Forda który stwierdził: *„Zabierzcie mi cały majątek, tylko zostawcie mi moich ludzi, a ja odbuduję wszystko”*. Zdanie to dobitnie pokazuje nam, jak cenny jest człowiek, który posiada odpowiednie cechy osobowości i umiejętności.

W tym, jak i w każdym innym projekcie znalazły się słabe ogniwa. Nie oznaczało to, że takie osoby należało od razu przekreślić. Przydzielenie im nowych zadań z jednoczesną kontrolą ich postępu, niejednokrotnie przyniosło wiele korzyści. Pokazało to przede wszystkim, jak wygląda praca nad projektem informatycznym oraz jak bardzo jest to trudne i skomplikowane przedsięwzięcie. Zostaliśmy zobligowani do połączenia kilku lat wiedzy i umiejętności, które przyswoiliśmy na uczelni z wielu przedmiotów, w celu realizacji **jednego** projektu. Specjalizacja ta pokazała jak na co dzień wygląda praca w korporacjach które zajmują się inżynierią oprogramowania. Jest to doświadczenie, które w 100% zaowocuje na naszej ścieżce kariery.

Stworzony system miał pokazać, jak według nas powinno wyglądać sprawne i efektywne zarządzanie wspólnotą. Zastosowanie informatyzacji zdecydowanie polepszy przepływ informacji wewnątrz wspólnoty oraz doda możliwość wpływu mieszkańców na jej zarządzanie. Dodaliśmy nowe funkcje, takie jak pomysły inwestycji, obsługę więcej niż jednej wspólnoty, przypisanie konta przez administratora dla lokatora wynajmującego mieszkanie oraz ankiety. Według naszych testów i rozeznania, trudno takich rzeczy szukać w konkurencyjnych systemach. Jako, iż jest to prototyp, według nas należałoby do finalnej wersji dodać:

- mobilną wersje strony;
- księgowanie płatności z rachunku bankowego;
- wysyłanie monitu o zaległościach na e-mail oraz sms;
- poprawić interfejs użytkowy i ogólny wygląd strony.

Mobilność - w dobie rozwoju urządzeń mobilnych wręcz wymagany jest wariant mobilny, który zapewniałby dostęp do informacji z dowolnego miejsca i urządzenia, które posiada dostęp do Internetu. Ze względu specyfiki budowy takiej strony należało by zadbać, aby była prosta, czytelna, i estetyczna oraz umożliwiałaby poprawne wyświetlanie dla każdej wielkości ekranu. Strona powinna zawierać nieprzewijające się, nierozbudowane elementy, a także duże widoczne przyciski.

Automatyzacja - dodanie automatycznego księgowania na pewno polepszyłoby komfort pracy użytkowników, a na pewno zmniejszyło koszty administracji poprzez obniżenie roboczogodzin pracownika. Powiadomienia o zaległościach w formie sms i e-maili rozwiązałoby problem z komunikacją i wiedzą mieszkańców o aktualnym stanie zadłużenia, bądź innych wydarzeniach mających miejsce w ich wspólnocie.

Wygląd - jest kwestią dyskusyjną i należałoby go zmodyfikować do aktualnie panujących trendów we wzornictwie, jak i jakości użytkowej. Powinno się także przeprowadzić testy użyteczności przez grupę kontrolną przyszłych użytkowników . W ten sposób sprawdzono by jak użytkownicy radzą sobie z obsługą systemu podczas wykonywania typowych zadań. Koszt takich badań jest znaczny i zależy od wielkości grupy oraz wykorzystanych narzędzi, jednakże dostarcza cennych informacji i wskazówek mogących poprawić zaufanie oraz zadowolenie przyszłego użytkownika aplikacji.

Rozwój o takie możliwości, w naszej ocenie, pozwoliłby na konkurowanie systemu na rynku komercyjnym. Są to rzeczy niezwykle ważne, bez których w dzisiejszych czasach trudno szukać miejsca na rynku.

9. Słownik pojęć

Administrator budynku – osoba odpowiedzialna za przegląd zgłoszeń, uprawniona do zmian stanów zgłoszeń oraz ich wystawiania.

Administrator systemu – osoba zajmująca się obsługą administracyjną systemu, posiada uprawnienia do wszelkich zmian w systemie.

Głosowanie – metoda podejmowania decyzji w zakresie funkcjonowania wspólnoty mieszkaniowej, przyjęcia bądź odrzucenia projektów inwestycyjnych. Głosowanie jest tajne/jawne, podejmowane większością bezwzględną.

Konto wspólnoty – konto bankowe, na którym gromadzony jest kapitał wspólnoty.

Konto lokatora – indywidualne konto bankowe, na które przelewane są środki pieniężne na pokrycie składek.

Księgowość – dział zajmujący się prowadzeniem finansów spółdzielni mieszkaniowej.

Lokator – osoba posiadająca prawo do wynajmowania danego lokalu na podstawie zawartej umowy najmu z właściciel mieszkania.

Mieszkanie – trwale wydzielony fragment nieruchomości – budynku mieszkalnego, służący do zaspokojenia potrzeb bytowych co najmniej jednej osoby.

Właściciel mieszkania – osoba posiadająca prawo własności do danej nieruchomości na podstawie aktu notarialnego.

Zarząd – organ o charakterze wykonawczym składający się z członków wspólnoty mieszkaniowej.

Zgłoszenie awarii – zawiadomienie przez lokatora bądź właściciela mieszkania o wystąpieniu awarii (problemu).

10. Bibliografia

1. **Sielski, Michał.** Spółdzielnia czy wspólnota. Co się bardziej opłaca? *Trojmiasto.pl.* [Online] 05 03 2013. [Zacytowano: 10 05 2015.] http://www.trojmiasto.pl/wiadomosci/Spoldzielnia-czy-wspolnota-Co-sie-bardziej-oplaca-n66835.html.

2. **Ewa, Wesołowska.** Biznes Onet. *Onet Biznes.* [Online] 17 04 2015. [Zacytowano: 12 05 2015.] http://biznes.onet.pl/wiadomosci/nieruchomosci/oplaty-za-wynajem-mieszkan-w-polsce-wsrod-najwyzszych-w-europie/slskb3.

3. Java . *Java About.* [Online] Oracle. [Zacytowano: 24 03 2015.] http://www.java.com/en/about/.

4. **SOFTWARE, TIOBE.** www. *TIOBE Index for March 2015.* [Online] TIOBE. [Zacytowano: 24 03 2015.] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

5. **Oracle.** *The History of Java Technology.* [Online] Oracle. [Zacytowano: 24 03 2015.] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

6. **Sonoo, Jaiswal.** JavaTPoint. *History Of Java.* [Online] [Zacytowano: 24 03 2015.] http://www.javatpoint.com/history-of-java.

7. **Wikimedia.** Wikibooks. *History of the Java™.* [Online] [Zacytowano: 21 03 2015.] http://en.wikibooks.org/wiki/Java_Programming/History.

8. **Lis, Marcin.** *Java. Ćwiczenia praktyczne, Wydanie III.* brak miejsca : Helion, 2011.

9. **Wikimedia Foundation.** C# (programming language). [Online] Wikipedia EN. [Zacytowano: 21 03 2015.] http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29.

10. **Eckel, Bruce.** *Thinking in Java, Edycja polska, wydanie IV.* brak miejsca : Helion, 2006.

11. **Cay S. Horstmann, Gary Cornell.** *Java. Podstawy; Wydanie IX.* brak miejsca : Helion, 2013.

12. **Wikimedia Foundation.** Wikipedia. *Java view technologies and frameworks.* [Online] [Zacytowano: 21 03 2015.] http://en.wikipedia.org/wiki/Java_view_technologies_and_frameworks.

13. **Maciejewski, Adam.** *JBoss Aplication Server.* [PDF] brak miejsca : MIMUW, 2006.

14. **RedHat.** [Online] [Zacytowano: 20 04 2015.] http://wildfly.org/about/.

15. **JavaTPoint.** JavaTPoint. *Hibernate Architecture.* [Online] [Zacytowano: 22 03 2015.] http://www.javatpoint.com/hibernate-architecture.

16. **Christian Bauer, Gavin King tłumaczenie Rafał Jońca.** *Hibernate in Action.* brak miejsca : Helion, 2007-07-11.

17. **Hibernate.** Hibernate Documentation. [Online] [Zacytowano: 23 03 2015.] http://hibernate.org/orm/documentation/getting-started/.

18. How To Do In Java. [Online] [Zacytowano: 24 03 2015.] http://howtodoinjava.com/2013/07/22/5-popular-java-development-frameworks/.

19. Spring. *Web MVC framework.* [Online] [Zacytowano: 11 04 2015.] http://docs.spring.io/.

20. **Spring.** Spring Security . [Online] [Zacytowano: 22 03 2015.] http://projects.spring.io/spring-security/.

21. **Project, Code.** *Getting Started Spring Security.* [Online] [Zacytowano: 21 03 2015.] http://www.codeproject.com/Articles/253901/Getting-Started-Spring-Security.

22. **Scarioni, Carlo.** *Pro Spring Security, Apress,.*

23. iOS Developer Library. *Model-View-Controller.* [Online] [Zacytowano: 09 04 2015.] https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

24. Bootstrap. *Getting Startet.* [Online] [Zacytowano: 26 03 2015.] http://getbootstrap.com/2.3.2/getting-started.html#examples.

25. EN Wikipedia. *Bootstrap (front-end framework).* [Online] [Zacytowano: 24 02 2015.] http://en.wikipedia.org/wiki/Bootstrap_%28front-end_framework%29.

26. **Bootstrap.** Base Css. [Online] [Zacytowano: 25 03 2015.] http://getbootstrap.com/2.3.2/base-css.html#typography.

27. **Boostrap.** Components. [Online] http://getbootstrap.com/2.3.2/components.html.

28. **WrapMarket LLC.** WrapBootstrap. *Boostrap Themes & Templates.* [Online] [Zacytowano: 26 03 2015.] http://wrapbootstrap.com/.

29. **Apache Software Fundation.** Apache Maven . *Project object model (POM).* [Online] http://maven.apache.org/index.html.

30. **FreeMarker project.** [Online] [Zacytowano: 12 05 2015.] http://freemarker.org/.

31. **Cernosek, Gary.** IBM developerWorks. *A brief history of Eclipse.* [Online] [Zacytowano: 02 04 2015.] http://www.ibm.com/developerworks/rational/library/nov05/cernosek/.

32. **Veys, Chris Laffra&Nick.** Eclipse Wiki. *FAQ Where did Eclipse come from?* [Online] [Zacytowano: 02 04 2015.] http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F.

33. Eclipse History. *About the Eclipse Foundation.* [Online] [Zacytowano: 02 04 2015.] http://www.eclipse.org/org/#history.

34. Centrum Doskonałości NIWA. *Jak system Redmine wspiera realizacje projektów.* [Online] [Zacytowano: 09 04 2015.] https://projects.os.niwa.gda.pl.

35. Redmine. [Online] [Zacytowano: 10 04 2015.] http://www.redmine.org/.

36. **Dąbrowski, Włodzimierz.** Budowa i wytwarzanie oprogramowania. *Wykłady - Cykle życiowe oprogramowania.* [Online] [Zacytowano: 14 05 2015.] http://edu.pjwstk.edu.pl/wyklady/byt/scb/start/index.html.

37. **R.K Wysocki, Rudd McGary.** *Efektywne zarządzanie projektami. Wydanie III.* brak miejsca : One press, 2005.08.23.

38. **Szymczak, Iwona.** *Wspólnota Mieszkaniowa.* brak miejsca : Wolters Kluwer, 2013.

39. **Hemrajani, Anil.** *Java. Tworzenie aplikacji sieciowych za pomoca Springa, Hibernate i Eclipsa.* brak miejsca : Helion, 2007.

40. **Bończak-Kucharczyk, Ewa.** *Zarządzanie nieruchomościami mieszkalnymi.* brak miejsca : Wolters Kluwer, 2014.

41. Dziennik Ustaw. *Nr 85 poz.388 USTAWA o własności lokali.* 24 czerwiec 1994.

42. **Apache Software Fundation.** Apache Maven. [Online] [Zacytowano: 18 04 2015.] https://maven.apache.org/.

11. Wykaz tabel

12. Wykaz rysunków

13. Wykaz listingów

Załącznik 1

Załącznik zawiera wszystkie tabele testowe fazy CRUD (112) powstałe podczas testowania aplikacji.

Tabela 22. CRUD obiekt uchwały. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Uchwały”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Wspólnota" a następnie wybierz "Uchwały"	
2	Wybierz „+„ oznaczający dodanie nowej uchwały	
3	Wybierz "Zapisz" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Zapisz"	
6	Wybierz ponownie „+„ oznaczający dodanie nowej uchwały	„Bad Request”
7	Wybierz z menu "Wspólnota" następnie "Uchwały"	
8	W panelu uchwał wybierz "Ołówek" oznaczający edycję uchwały	
9	W panelu uchwał wybierz "X" oznaczający usunięcie uchwały	
Oczekiwany rezultat	1.Dodanie uchwały 2.Wyświetlenie uchwały 3.Edycja uchwały 4.Usunięcie uchwały	
Uzyskany rezultat	Niezgodny z oczekiwaniem	
Klasa błędu	1	
Wynik testu	Zgodny po dokonaniu poprawek	

Tabela 23. CRUD obiekt płatności. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Płatności”
Warunki początkowe	Zalogowany użytkownik z prawami administratora
Realizacja scenariusza testowego	

Krok	Opis	Uwagi
1	Z lewego menu kliknij "Wspólnota" a następnie wybierz "Lista Płatności"	
2	Wybierz "Dodaj" bez wpisywania danych	
3	Wybierz "Anuluj"	Brak powrotu
4	Wybierz "Dodaj"	
5	Wypełnij dostępne pola i naciśnij "Dodaj Płatność"	
6	Ponów krok 5	Można 2 x dodać identyczna płatność
7	Wybierz "Szczegóły"	Brak powrotu
8	Wybierz "Edycja"	Brak powrotu
9	Wybierz "Usuń"	
Oczekiwany rezultat	1.Dodanie płatności 2.Edycja płatności 3.Usunięci płatności 4.Przeglądanie płatności	
Uzyskany rezultat	zgodny po poprawkach	
Klasa błędu	2	
Wynik testu	Zgodny po uwzględnieniu poprawek	

Tabela 24. CRUD obiektu raporty. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Raporty”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Wspólnota" a następnie wybierz "Raporty"	
2	Wybierz „+„ oznaczający dodanie nowego raportu	
3	Wybierz "Dodaj raport" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj raport"	
6	Wybierz ponownie „+„ oznaczający dodanie nowego raportu	
7	Wybierz z menu "Wspólnota" następnie "Raporty"	
8	W panelu raportów wybierz "Ołówek" oznaczający edycję raportu	

9	W panelu raportów wybierz "X" oznaczający usunięcie raportu	
Oczekiwany rezultat	1.Dodanie raportu 2.Wyświetlenie raportów 3. Edycja raportu 4. Usunięcie raportu	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	
Wynik testu	Zgodny	

Tabela 25. CRUD obiektu awarie. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Awarie”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Wspólnota" a następnie wybierz "Awarie"	
2	Wybierz „+„ oznaczający dodanie nowej Awarii	
3	Wybierz "Dodaj" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj"	
6	Wybierz ponownie „+„ oznaczający dodanie nowej awarii	
7	Wybierz z menu "Wspólnota" następnie "Awarie"	
8	W panelu awarii wybierz "Ołówek" oznaczający edycję awarii	
9	W panelu awarii wybierz "X" oznaczający usunięcie awarii	
Oczekiwany rezultat	1.Dodanie awarii 2.Wyświetlenie awarii 3.Edycja awarii 4.Usunięcie awarii	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	
Wynik testu	Zgodny	

Tabela 26. CRUD obiektu ankiety. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Ankiety”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Wspólnota" a następnie wybierz "Ankiety"	
2	Wybierz „+„ oznaczający dodanie nowej ankiety	
3	Wybierz "Dodaj ankietę" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj ankietę"	
6	Wybierz ponownie „+„ oznaczający dodanie nowej ankiety	
7	Wybierz z menu "Wspólnota" następnie "Ankiety"	
8	W panelu awarii wybierz "Ołówek" oznaczający edycję ankiety	
9	W panelu awarii wybierz "X" oznaczający usunięcie ankiety	
Oczekiwany rezultat	1.Dodanie ankiety 2.Wyświetlenie ankiety 3. Edycja ankiety 4. Usunięcie ankiety	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	
Wynik testu	Zgodny	

Tabela 27. CRUD obiektu inwestycje. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Inwestycje”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z lewego menu kliknij "Inwestycje" a następnie wybierz "Dodaj inwestycje"	

2	Wybierz "Dodaj" bez wpisywania danych	
3	Wypełnij dostępne pola	1. Pola "Status" nie rozwiązano za pomocą drop listy
4	Wybierz "Dodaj"	Internal Server Error
5	Kliknij inwestycje a następnie wybierz "Przeglądaj inwestycje"	
6	Wybierz "Szczegóły" inwestycji	Internal Server Error
7	Wybierz "Edycja" dokonaj zmian i zapisz	
8	Wybierz "Usuń"	Not Found
Oczekiwany rezultat	1. Dodanie Inwestycji 2.Edycja Inwestycji 3. Usuniecie Inwestycji 4. Przeglądanie Inwestycji	
Uzyskany rezultat	negatywny	
Klasa błędu	1	
Wynik testu	Wymaga pilnej naprawy	

Tabela 28. CRUD obiektu projekty. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Projekty”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Inwestycje" a następnie wybierz "Projekty"	
2	Wybierz „+„ oznaczający dodanie nowego projektu	
3	Wybierz "Zgłoś projekt" bez wpisywania danych	Możliwość dodania projektu bez wypełnienia pola "Opis"
4	Wypełnij dostępne pola	
5	Wybierz "Zgłoś projekt"	
6	Wybierz ponownie „+„ oznaczający dodanie nowej ankiety	
7	Wybierz z menu "Inwestycje" następnie "Projekty"	
8	W panelu projektów wybierz "Ołówek" oznaczający edycję projektu	

9	W panelu projektów wybierz "X" oznaczający usunięcie projektu	
Oczekiwany rezultat	1.Dodanie ankiety 2.Wyświetlenie ankiety 3. Edycja ankiety 4. Usunięcie ankiety	
Uzyskany rezultat	Niezgodny z oczekiwaniem	
Klasa błędu	2	
Wynik testu	Zgodny po dokonaniu poprawek	

Tabela 29. CRUD obiektu użytkownicy. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Użytkownicy”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Panel administracyjny" a następnie wybierz "Zarządzanie użytkownikami"	
2	Wybierz „+„ oznaczający dodanie nowego użytkownika	
3	Wybierz "Dodaj użytkownika" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	1.1.1.1. Wybierz "Dodaj użytkownika"	
6	Wybierz ponownie „+„ oznaczający dodanie nowej ankiety	
7	Wybierz z menu "Panel administracyjny" następnie "Zarządzanie użytkownikami"	
8	W panelu użytkowników wybierz "Ołówek" oznaczający edycję użytkownika.	
9	W użytkowników wybierz "X" oznaczający usunięcie użytkownika	
Oczekiwany rezultat	1.Dodanie użytkownika 2. Wyświetlenie użytkownika 3. Edycja użytkownika 4. Usunięcie użytkownika	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	

Wynik testu	Zgodny

Tabela 30. CRUD obiektu budynki. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Budynki”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Panel administracyjny" a następnie wybierz "Budynki"	
2	Wybierz „+„ oznaczający dodanie nowego budynku	
3	Wybierz "Dodaj budynek" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj budynek"	
6	Wybierz ponownie „+„ oznaczający dodanie nowego budynku	
7	Wybierz z menu "Panel administracyjny" następnie "Budynki"	
8	W panelu budynków wybierz "Ołówek" oznaczający edycję budynku	
9	W panelu budynków wybierz "X" oznaczający usunięcie budynku	
Oczekiwany rezultat	1.Dodanie budynku 2. Wyświetlenie budynku 3. Edycja budynku 4. Usunięcie budynku	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	
Wynik testu	Zgodny	

Tabela 31. CRUD obiektu mieszkania. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Mieszkania”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**

1	Z górnego menu kliknij "Panel administracyjny" a następnie wybierz "Mieszkania"	
2	Wybierz „+„ oznaczający dodanie nowego mieszkania	
3	Wybierz "Dodaj mieszkanie" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj mieszkanie"	
6	Wybierz ponownie „+„ oznaczający dodanie nowego mieszkania	
7	Wybierz z menu "Panel administracyjny" następnie "Mieszkania"	
8	W panelu mieszkań wybierz "Ołówek" oznaczający edycję mieszkania	
9	W panelu mieszkań wybierz "X" oznaczający usunięcie mieszkania	
Oczekiwany rezultat	1.Dodanie mieszkania 2. Wyświetlenie mieszkania 3. Edycja mieszkania 4. Usunięcie mieszkania	
Uzyskany rezultat	Zgodny z oczekiwaniem	
Klasa błędu	0	
Wynik testu	Zgodny	

Tabela 32. CRUD obiektu liczniki. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Tworzenie, edycja i usuwanie obiektu „Liczniki”	
Warunki początkowe	Zalogowany użytkownik z prawami administratora	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z górnego menu kliknij "Panel administracyjny" a następnie wybierz "Przeglądaj Liczniki"	
2	Wybierz „+„ oznaczający dodanie nowego licznika	
3	Wybierz "Dodaj licznik" bez wpisywania danych	
4	Wypełnij dostępne pola	
5	Wybierz "Dodaj licznik"	
6	Wybierz ponownie „+„ oznaczający dodanie nowego licznika	

7	Wybierz z menu "Panel administracyjny" następnie "Przeglądaj liczniki"	
8	W panelu liczników wybierz "Ołówek" oznaczający edycję licznika	"Dostęp zabroniony"
9	W panelu liczników wybierz "X" oznaczający usunięcie licznika	
Oczekiwany rezultat	1.Dodanie licznika 2. Wyświetlenie licznika 3. Edycja licznika 4. Usunięcie licznika	
Uzyskany rezultat	Niezgodny z oczekiwaniem	
Klasa błędu	1	
Wynik testu	Zgodny po dokonaniu poprawek	

Załącznik 2

Załącznik zawiera wszystkie pozostałe tabele z drugiej fazy testów (115).

Tabela 33. Obiekt płatności. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Płatności	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Płatności", a na końcu "+" oznaczający dodanie nowej uchwały.	
2	Wypełnić następujące pola: • Tytuł płatności • Płatnik • Kwota • Komentarz • Mieszkanie	
3	Wybierz "Dodaj"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Płatności w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Płatności w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 34. Obiekt raporty. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Raporty	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Raporty", a na końcu "+" oznaczający dodanie nowego raportu.	
2	Wypełnić następujące pola: • Wybranie wspólnoty • Tytuł • Treść	
3	Wybierz "Dodaj raport"	

Oczekiwany rezultat	Zarejestrowano nowy obiekt Raporty w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Raporty w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 35. Obiekt awarie. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Awarie	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Awarie", a na końcu "+" oznaczający dodanie nowej awarii	
2	Wypełnić następujące pola: • Wybranie wspólnoty • Opis awarii • Wybranie kategorii • Data wystąpienia awarii	
3	Wybierz "Dodaj"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Awarie w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Awarie w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 36. Obiekt ankiety. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Ankiety	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Wspólnota", następnie wybierz "Ankiety", a na końcu "+" oznaczający dodanie nowej ankiety	

2	Wypełnić następujące pola • Wybranie wspólnoty • Data aktywności • Treść zapytania • Odpowiedzi	
3	Wybierz "Dodaj ankietę"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Ankiety w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Ankiety w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 37. Obiekt inwestycje. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Inwestycje	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Inwestycje", następnie wybierz "Lista inwestycji", a na końcu "+" oznaczający dodanie nowej inwestycji.	
2	Wypełnić następujące pola: • Wybierz wspólnotę • Nazwa inwestycji • Opis • Status • Wartość	
3	Wybierz "Dodaj"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Inwestycje w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Inwestycje w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 38. Obiekt projekt. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Projekty	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**

1	Z menu wybierz "Inwestycje", następnie wybierz "Projekty", a na końcu "+" oznaczający dodanie nowego projektu	
2	Wypełnić następujące pola: • Opis	
3	Wybierz "Zgłoś projekt"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Projekty w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Projekty w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 39. Obiekt operacja finansowa. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Operacje finansowe	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Inwestycje", następnie wybierz "Operacje finansowe", a na końcu "+" oznaczający dodanie nowej operacji finansowej.	
2	Wypełnić następujące pola: • Kwota płatności • Opis • Inwestycja	
3	Wybierz "Zapisz operacje"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Operacje finansowe w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Operacje finansowe w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 40. Obiekt użytkownicy. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Użytkownicy
Warunki początkowe	Zalogowany użytkownik
Realizacja scenariusza testowego	

Krok	Opis	Uwagi
1	Z menu wybierz "Panel administracyjny", następnie wybierz "Zarządzanie użytkownikami", a na końcu "+" oznaczający dodanie nowego użytkownika	
2	Wypełnić następujące pola: • Login • Hasło • Potwierdź hasło • Imię • Nazwisko • PESEL • Numer telefonu • Numer licencji (zarządca) • Ulica • Numer budynku • Numer lokalu • Kod pocztowy • Wybranie województwa Miejscowość	
3	Wybierz "Dodaj użytkownika"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Użytkownicy w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Użytkownicy w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 41. Obiekt budynki. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Budynki	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Panel administracyjny", następnie wybierz "Budynki", a na końcu "+" oznaczający dodanie nowego budynku	

2	Wypełnić następujące pola: • Ulica • Numer budynku • Kod pocztowy • Wybranie województwa • Miejscowość • Wybranie przydzielenia do wspólnoty	
3	Wybierz "Dodaj budynek"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Budynki w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Budynki w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Tabela 42. Obiekt mieszkanie. ***Źródło: Opracowanie własne.***

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Mieszkania	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Panel administracyjny", następnie wybierz "Mieszkania", a na końcu "+" oznaczający dodanie nowego mieszkania	
2	Wypełnić następujące pola: • Numer mieszkania • Piętro • Metraż • Liczba mieszkańców • Wybranie przydzielenia do budynku	1. Po uzupełnieniu pola "Metraż" samymi literami lub wykroczeniem poza określony limit metrażu np. wpiszemy "555" występuje błąd "Niewłaściwe parametry żądania". 2. Po uzupełnieniu pola "Liczba mieszkańców" samymi literami występuje błąd "Niewłaściwe parametry żądania".
3	Wybierz "Dodaj mieszkanie"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Mieszkanie w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Mieszkanie w bazie danych	
Klasa błędu	1	
Wynik testu	Wymaga pilnej naprawy	

Tabela 43. Obiekt liczniki. *Źródło: Opracowanie własne.*

Nazwa scenariusza	Weryfikacja wprowadzenia danych - obiekt Liczniki	
Warunki początkowe	Zalogowany użytkownik	
Realizacja scenariusza testowego		
Krok	**Opis**	**Uwagi**
1	Z menu wybierz "Panel administracyjny", następnie wybierz "Mieszkania", a na końcu "+" oznaczający dodanie nowego licznika	
2	Wypełnić następujące pola: • Typ • Numer seryjny • Wybranie mieszkania	
3	Wybierz "Dodaj licznik"	
Oczekiwany rezultat	Zarejestrowano nowy obiekt Licznik w bazie danych	
Uzyskany rezultat	Zarejestrowano nowy obiekt Licznik w bazie danych	
Klasa błędu	0	
Wynik testu	Pozytywny	

Załącznik 3

Załącznik zawiera wszystkie scenariusze powstałe w fazie projektowania aplikacji (84).

Tabela 44. Scenariusz przeglądania ogłoszeń. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądanie ogłoszeń**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. W systemie istnieje co najmniej jedno ogłoszenie	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie ogłoszeń w Menu
	2	System wyświetla kategorie ogłoszeń
	3	Lokator wybiera interesującą go kategorię ogłoszeń
	4	System wyświetla ogłoszenia posortowane malejąco względem daty, należące do wybranej przez lokatora kategorii
	5	Lokator wybiera ogłoszenie
	6	System wyświetla szczegóły wybranego ogłoszenia
	7	Lokator przegląda szczegóły ogłoszenia. Koniec przypadku użycia

Tabela 45. Scenariusz przeglądania uchwał. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądanie dziennika uchwał**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. W systemie istnieje co najmniej jedna uchwała	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze przeglądanie dziennika uchwał w Menu

	2	System wyświetla listę uchwał posortowaną, malejącą względem daty
	3	Lokator wybiera interesującą go uchwałę
	4	System wyświetla szczegóły całościowy opis wybranej uchwały
	5	Lokator przegląda uchwałę. Koniec przypadku użycia

Tabela 46. Scenariusz deklaracji stanu licznika. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Deklaracja stanu licznika**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora	
Warunki końcowe	W systemie zostaje poprawnie zapisany nowy stan licznika	
Aktor pierwszoplanowy	Lokator	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze Liczniki w Menu
	2	System wyświetla widok stanów liczników mieszkania lokatora
	3	Lokator wybiera dodanie nowego stanu licznika
	4	System prosi o wybranie rodzaju licznika, dla którego ma zostać zapisany stan
	5	Lokator wybiera rodzaj licznika
	6	System wyświetla formularz dodawania stanu licznika
	7	Lokator wypełnia formularz i wybiera przycisk Zapisz. Koniec przypadku użycia

Tabela 47. Scenariusz edycji stanu licznika. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Edycja stanu licznika**
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. W systemie istnieje co najmniej jedna deklaracja stanu licznika
Warunki końcowe	W systemie zostaje poprawnie nadpisany stan licznika
Aktor pierwszoplanowy	Lokator

Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy lokator wybierze Liczniki w Menu
	2	System wyświetla widok stanów liczników mieszkania lokatora
	3	Lokator wybiera pożądany licznik do edycji
	4	System wyświetla formularz edycji stanu licznika
	5	Lokator wypełnia formularz i wybiera przycisk Zapisz. Koniec przypadku użycia.

Tabela 48. Scenariusz przeglądania kont lokatorów. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądanie kont lokatorów**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. W systemie istnieje co najmniej jedno konto lokatora przypisane do mieszkania danego właściciela.	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Lokatorzy w Menu
	2	System wyświetla wszystkie konta lokatorów powiązane z mieszkaniem właściciela
	3	Właściciel wybiera konto lokatora
	4	System wyświetla szczegółowe informacje o wybranym koncie lokatora
	5	Właściciel ogląda szczegóły wybranego konta. Koniec przypadku użycia

Tabela 49. Scenariusz tworzenia konta lokatora. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Tworzenie konta lokatora**
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania
Warunki końcowe	W systemie zostaje poprawnie zapisane nowe konto lokatora

Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Lokatorzy w Menu
	2	System wyświetla wszystkie konta lokatorów powiązane z mieszkaniem właściciela
	3	Właściciel wybiera Dodanie lokatora
	4	System wyświetla formularz dodawania nowego konta lokatora
	5	Właściciel wypełnia wszystkie wymagane pola w formularzu w wybiera Dodaj
	6	System zapisuje nowe konto lokatora. Koniec przypadku użycia

Tabela 50. Scenariusz edycji konta lokatora. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Edycja konta lokatorów**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. W systemie istnieje co najmniej jedno konto lokatora przypisane do mieszkania danego właściciela	
Warunki końcowe	W systemie zostają poprawnie nadpisane zmiany w koncie lokatora	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Lokatorzy w Menu
	2	System wyświetla wszystkie konta lokatorów powiązane z mieszkaniem właściciela
	3	Właściciel wybiera konto lokatora
	4	System wyświetla szczegółowe informacje o wybranym koncie lokatora
	5	Właściciel wybiera Edytuj
	6	System wyświetla formularz edycji konta lokatora
	7	Właściciel nanosi zmiany na formularzu konta lokatora i wybiera przycisk Zapisz

	8	System nadpisuje zmiany w danym koncie lokatora. Koniec przypadku użycia.

Tabela 51. Scenariusz usuwania konta lokatora. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Usuwanie konta lokatora**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. W systemie istnieje co najmniej jedno konto lokatora przypisane do mieszkania danego właściciela	
Warunki końcowe	System poprawnie usuwa konto lokatora z systemu	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Lokatorzy w Menu
	2	System wyświetla wszystkie konta lokatorów powiązane z mieszkaniem właściciela
	3	Właściciel wybiera konto lokatora
	4	System wyświetla szczegółowe informacje o wybranym koncie lokatora
	5	Właściciel wybiera Usuń
	6	System wyświetla komunikat z prośbą o potwierdzenie wykonania operacji usuwania
	7	Właściciel wybiera przycisk Tak
	8	System usuwa wybrane konto. Koniec przypadku użycia.

Tabela 52. Scenariusz podglądu konta finansowego wspólnoty. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądanie konta wspólnoty**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. Aktor należy do wspólnoty mieszkaniowej danego konta	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**

	1	Przypadek zaczyna się, gdy właściciel wybierze Wspólnota w Menu
	2	System wyświetla informacje o wspólnocie, do której należy właściciel
	3	Właściciel wybiera przycisk Finanse
	4	System wyświetla szczegółowe informacje o koncie finansowym wspólnoty
	5	Właściciel ogląda szczegóły wybranego konta finansowego wspólnoty. Koniec przypadku użycia

Tabela 53. Scenariusz sprawdzania stanu konta mieszkania. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Sprawdzanie stanu konta mieszkania**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania	
Warunki końcowe	Nie zostaje zapisana żadna nowa informacja w systemie	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Mieszkanie w Menu
	2	System wyświetla informacje o mieszkaniu właściciela
	3	Właściciel wybiera przycisk Stan Konta
	4	System wyświetla szczegółowe informacje o stanie konta mieszkania
	5	Właściciel ogląda szczegóły stanu konta. Koniec przypadku użycia

Tabela 54. Scenariusz edycji deklaracji osób. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Edycja deklaracji osób zajmujących mieszkanie**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania	
Warunki końcowe	W systemie zostaje zapisana nowa liczba osób	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**

	1	Przypadek zaczyna się, gdy właściciel wybierze Mieszkanie w Menu
	2	System wyświetla informacje o mieszkaniu właściciela
	3	Właściciel wybiera przycisk Deklaracja Osób w Mieszkaniu
	4	System wyświetla listę osób zajmujących dane mieszkanie
	5	Właściciel nanosi zmiany (dodaje, usuwa osoby) na liście osób zajmujących mieszkania i wybiera przycisk Zapisz. Koniec przypadku użycia

Tabela 55. Scenariusz podglądu kalendarza. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądanie kalendarza spotkań**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. W systemie jest zaznaczone co najmniej jedno spotkanie	
Warunki końcowe	brak	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze Kalendarz w Menu
	2	System wyświetla kalendarz z zaznaczonymi spotkaniami organizowanymi w ramach danej wspólnoty
	3	Właściciel wybiera dowolne spotkanie z kalendarza
	4	System wyświetla szczegóły spotkania odbywającego się w danym dniu
	5	Właściciel ogląda szczegóły wybranego spotkania. Koniec przypadku użycia

Tabela 56. Scenariusz głosowania w ankiecie. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Głosowanie w ankiecie**
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia właściciela mieszkania. W systemie jest uruchomiona co najmniej jedna ankieta. Aktor nie oddawał do tej pory głosu w danej ankiecie.
Warunki końcowe	Punkt zostaje poprawnie zapisany do agendy danego spotkania

Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze przeglądanie projektów inwestycji w Menu
	2	System wyświetla widok z pomysłami inwestycji
	3	Właściciel wybiera dowolny projekt inwestycji
	4	System wyświetla opis danego projektu inwestycji
	5	Właściciel oddaje głos na poparcie bądź odrzucenie danej inwestycji
	6	System zapisuje głos właściciela. Koniec przypadku użycia

Tabela 57. Scenariusz akceptacji pomysłów inwestycji lokatora. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Rozpatrywanie pomysłów inwestycyjnych lokatora**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia lokatora. Lokator mieszkania właściciela dodał pomysł na inwestycje w wspólnocie.	
Warunki końcowe	System zapisuje decyzję właściciela mieszkania w odniesieniu do pomysłu inwestycyjnego zgłoszonego przez lokatora	
Aktor pierwszoplanowy	Właściciel mieszkania	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy właściciel wybierze przycisk Lokatorzy w menu
	2	System wyświetla informacje o kontach lokatorów mieszkania właściciela
	3	Właściciel wybiera przycisk Do Zaakceptowania
	4	System wyświetla wszystkie nierozważone propozycje inwestycji lokatorów mieszkania właściciela
	5	Właściciel wybiera propozycję inwestycji
	6	System wyświetla wybraną propozycję inwestycji
	7	Właściciel akceptuje propozycję lokatora

	8	System upublicznia daną propozycję do widoku ogólnego propozycji inwestycji. Koniec przypadku użycia
Alternatywne przepływy zdarzeń	**Krok**	**Akcja**
	7a	Właściciel odrzuca propozycję lokatora
	8a	System usuwa daną propozycję lokatora. Koniec przypadku użycia

Tabela 58. Scenariusz dodawania ogłoszeń. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Dodawanie ogłoszeń**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu. W obszarze wpisu powinna być możliwość formatowania tekstu oraz wklejania do treści obrazków	
Warunki końcowe	Ogłoszenie zostaje poprawnie zapisane w systemie	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza zamieścić ogłoszenie i wybiera zakładkę Wyświetl ogłoszenia
	2	Wybiera w systemie opcję Dodaj
	3	System wyświetla formularz
	4	Członek zarządu wpisuje temat i treść ogłoszenia
	5	Członek zarządu klika na przycisku Dodaj ogłoszenie
	6	System prosi o potwierdzenie
	7	Członek zarządu potwierdza
	8	System zapisuje ogłoszenie
Alternatywne przepływy zdarzeń	5a	1. Członek zarządu wraca do poprzedniego ekranu bez zapisu 2. System zamyka formularz
	5b	1. Członek zarządu wybiera Powróć do edycji 2. System powraca do edycji formularza

Tabela 59. Scenariusz propozycji inwestycyjnych. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Przejrzyj propozycje inwestycyjne mieszkańców**

Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu. W systemie istnieje co najmniej jedna propozycja inwestycyjna	
Warunki końcowe	Propozycja inwestycji zostaje zapisana w systemie jako przeznaczona do realizacji	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza przejrzeć propozycje inwestycyjne mieszkańców, wybiera w systemie opcję Przeglądaj projekty
	2	System wyświetla listę
	3	Członek zarządu wybiera pozycję z listy
	4	System wyświetla szczegółowe informacje na temat wniosku
	5	Członek zarządu wybiera opcję Zapisz i zakończ
	6	System prosi o potwierdzenie
	7	Członek zarządu potwierdza
	8	System zapisuje wniosek w archiwum pomysłów możliwych do realizacji
Alternatywne przepływy zdarzeń	5a	1. Członek zarządu wybiera opcję Odeślij do uzupełnienia 2. System wyświetla formularz, w którym można wpisać uwagi 3. Członek zarządu wpisuje uwagi i wybiera opcję Wyślij 4. System prosi o potwierdzenie, członek zarządu potwierdza 5. System przekazuje do nadawcy wniosek wraz z komentarzem członka zarządu
	5b	1. Członek zarządu wybiera opcję Odrzuć pomysł 2. System wyświetla formularz, w którym można wpisać uwagi 3. Członek zarządu wpisuje uwagi i wybiera opcję Wyślij 4. System prosi o potwierdzenie 5. Członek zarządu potwierdza 6. System zwraca do nadawcy wniosek wraz z komentarzem członka zarządu

Tabela 60. Scenariusz podglądu wyników ankiety. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przejrzyj wyniki ankiety**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu. W systemie istnieje co najmniej jedna ankieta	
Warunki końcowe	Członek zarządu zapoznaje się z wynikami ankiet. System nie zapisuje żadnych nowych zmian	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	1. Przypadek zaczyna się, członek zarządu zamierza przejrzeć ankiety, wybiera w systemie opcję Wyświetl ankiety 2. System wyświetla szczegółowe informacje na temat ankiety i jej wyniku 3. Członek zarządu wybiera opcję Zakończ 4. System prosi o potwierdzenie 5. Członek zarządu potwierdza 6. System powraca do listy 7. Członek zarządu wybiera opcję Zakończ 8. System prosi o potwierdzenie 9. Członek zarządu potwierdza System zamyka listę
	2	System wyświetla listę
	3	Członek zarządu wybiera pozycję z listy
	4	System wyświetla szczegółowe informacje na temat ankiety i jej wyniku
	5	Członek zarządu wybiera opcję Zakończ
	6	System prosi o potwierdzenie
	7	Członek zarządu potwierdza
	8	System zamyka listę
Alternatywne przepływy zdarzeń		

Tabela 61. Scenariusz dodawania ankiet. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Zamieść Ankietę**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu	
Warunki końcowe	System zapisuje nową ankietę	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza zamieścić ankietę.
	2	Członek zarządu wybiera w systemie opcję Wyświetl ankiety, a następnie Dodaj ankietę
	3	System wyświetla formularz
	4	Członek zarządu wpisuje temat i treść ankiety
	5	Członek zarządu wybiera opcję Dodaj ankietę
	6	System prosi o potwierdzenie
	7	Członek zarządu potwierdza
	8	System zapisuje ankietę
Alternatywne przepływy zdarzeń	5a	1. Członek zarządu wybiera opcję cofnięcia do poprzedniego ekranu 2. System zamyka formularz
	7a	1. Członek zarządu wybiera Powróć do edycji 2. System powraca do edycji formularza

Tabela 62. Scenariusz dodawania zdarzeń w kalendarzu. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Wprowadź zdarzenie do kalendarza**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu	
Warunki końcowe	System zapisuje nowe zdarzenie	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza wprowadzić zdarzenie do kalendarza

	2	Członek zarządu wybiera w systemie na ekranie Kalendarz opcję Dodaj zdarzenie do kalendarza
	3	System wyświetla kalendarz wraz z możliwością wybrania daty i godziny
	4	Członek zarządu wpisuje temat i treść ogłoszenia, wybiera datę i ewentualnie godzinę oraz miejsce wydarzenia
	5	Członek zarządu wybiera opcję Zapisz
	6	System prosi o potwierdzenie
	7	Członek zarządu potwierdza
	8	System zapisuje i publikuje informację o zdarzeniu
Alternatywne przepływy zdarzeń	5a	1. Członek zarządu wybiera opcję dodaj dokument do spotkania 2. System wyświetla panel wyboru źródła pliku 3. Członek zarządu wybiera i potwierdza 4. System prosi o potwierdzenie zamiaru 5. Członek zarządu potwierdza 6. System zapisuje dokument do zdarzenia
	7a	1. Członek zarządu wybiera Powróć do edycji 2. System powraca do edycji formularza

Tabela 63. Scenariusz dodawania informacji o inwestycji. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Wprowadź informację o inwestycji**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu	
Warunki końcowe	System zapisuje nową informację o inwestycji	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza wprowadzić informację o inwestycji
	2	Członek zarządu wybiera w systemie zakładkę Przeglądaj inwestycje
	3	System wyświetla listę wprowadzonych inwestycji

	4	Członek zarządu wybiera w systemie opcję Dodaj
	5	System wyświetla formularz
	6	Członek zarządu wpisuje temat i treść informacji, a następnie wybiera opcję Dodaj inwestycję
	7	System prosi o potwierdzenie
	8	Członek zarządu potwierdza
	8	System zapisuje i publikuje informację
Alternatywne przepływy zdarzeń	5a	1. Członek zarządu wybiera opcję Dodaj dokument do spotkania 2. System wyświetla panel wyboru źródła pliku 3. Członek zarządu wybiera i potwierdza 4. System prosi o potwierdzenie zamiaru 5. Członek zarządu potwierdza 6. System zapisuje dokument do zdarzenia
	7a	1. Członek zarządu wybiera Powróć do edycji 2. System powraca do edycji formularza

Tabela 64. Scenariusz edycji informacji o inwestycji. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Edytuj informację o inwestycji**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia członka zarządu. W systemie jest co najmniej jedna informacja o inwestycji	
Warunki końcowe	System zapisuje nową informację o inwestycji	
Aktor pierwszoplanowy	Członek zarządu	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, członek zarządu zamierza zmienić informację o inwestycji
	2	Członek zarządu wybiera w systemie zakładkę Przeglądaj inwestycje
	3	System wyświetla listę wprowadzonych inwestycji
	4	Członek zarządu wybiera pozycję z listy

	5	System wyświetla szczegółowe informacje na temat inwestycji
	6	Członek zarządu wybiera opcję Edytuj
	7	System wyświetla formularz
	8	Członek zarządu wpisuje nowe informacje
	9	Członek zarządu wybiera opcję Zapisz
	10	System prosi o potwierdzenie
	11	System zapisuje zmiany
Alternatywne przepływy zdarzeń		

Tabela 65. Scenariusz raportu administratora. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Dodaj wpis raportu administratora**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla	
Warunki końcowe	W systemie zostaje zapisany nowy wpis	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy z widoku Raport użytkownik wybiera opcję Dodaj wpis.
	2	System wyświetla okienko z formularzem
	3	Użytkownik wypełnia dane i zatwierdza wybierając opcję Zapisz
	4	System potwierdza zapisanie i wraca do strony z wpisami
Alternatywne przepływy zdarzeń		

Tabela 66. Scenariusz podglądu awarii. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Przeglądaj zgłoszenia awarii**
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla. W systemie istnieje co najmniej

	jedno zgłoszenie awarii	
Warunki końcowe	W systemie nie zostają zapisane żadne nowe informacje	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy z menu głównego użytkownik wybiera opcję Przeglądaj awarie
	2	System wyświetla listę zgłoszeń, na której widoczne są następujące informacje: a. Data awarii b. Opis c. Status
	3	Użytkownik wybiera Szczegóły
	4	System wyświetla okienko ze szczegółami awarii
Alternatywne przepływy zdarzeń		

Tabela 67. Scenariusz edycji informacji o awarii. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Edytuj informację o awarii**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla. W systemie istnieje co najmniej jedno zgłoszenie awarii	
Warunki końcowe	W systemie zostaje zapisana zmiana informacji dotycząca awarii	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy z listy awarii użytkownik wybiera opcję Edytuj
	2	System wyświetla okno z następującymi polami: a. Data b. Opis c. Status
	3	Użytkownik wprowadza dane i wybiera opcję Zapisz
	4	System aktualizuje informację o awarii
Alternatywne przepływy		

zdarzeń		

Tabela 68. Scenariusz dodawania ogłoszeń. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Wstaw ogłoszenie**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla	
Warunki końcowe	W systemie zostaje zapisane nowe ogłoszenie	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Przypadek zaczyna się, gdy z menu głównego użytkownik wybiera opcję Wyświetl ogłoszenia, a następnie Dodaj
	2	System wyświetla okienko z formularzem z następującymi danymi do wypełnienia: a. Tytuł ogłoszenia b. Treść c. Kategoria (lista wyboru) d. Zdjęcie e. Priorytet
	3	Użytkownik wypełnia dane i zatwierdza wybierając opcję Zapisz
	4	System potwierdza zapisanie, zapisuje dane i wraca do strony głównej
Alternatywne przepływy zdarzeń		

Tabela 69. Scenariusz edycji ogłoszeń. *Źródło: Opracowanie własne.*

Nazwa przypadku użycia	**Edytuj ogłoszenie**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla. W systemie istnieje zapisane co najmniej jedno ogłoszenie	
Warunki końcowe	W systemie zostają zapisane nowe informacje dotyczące ogłoszenia	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Z widoku ogłoszeń użytkownik wybiera opcję Edytuj przy konkretnym ogłoszeniu

	2	System wyświetla okienko z formularzem z następującymi danymi do edycji: a. Tytuł ogłoszenia b. Treść c. Kategoria (lista wyboru) d. Zdjęcie e. Priorytet
	3	Użytkownik wypełnia dane i zatwierdza wybierając opcję Zapisz
	4	System potwierdza zapisanie, zapisuje dane i wraca do strony głównej
Alternatywne przepływy zdarzeń		

Tabela 70. Scenariusz edycji wpisu administratora. ***Źródło: Opracowanie własne.***

Nazwa przypadku użycia	**Edytuj wpis raportu administratora osiedla**	
Warunki wstępne	Aktor został poprawnie zalogowany do systemu i posiada uprawnienia administratora osiedla. W systemie istnieje zapisany co najmniej jeden wpis w raporcie	
Warunki końcowe	W systemie zostają zapisane nowe informacje dotyczące raportu	
Aktor pierwszoplanowy	Administrator osiedla	
Główny przepływ zdarzeń	**Krok**	**Akcja**
	1	Z widoku szczegółów wpisu użytkownik wybiera opcję Edytuj”
	2	System wyświetla okienko z formularzem
	3	Użytkownik wypełnia dane i zatwierdza wybierając opcję Zapisz
	4	System potwierdza zapisanie, zapisuje dane i wraca do strony głównej
Alternatywne przepływy zdarzeń		