

Wyższa Szkoła Technologii Informatycznych w Katowicach

Wydział Informatyki
Kierunek Informatyka

Adrian Leś

Nr albumu: 34150

Studia stacjonarne

Temat pracy

System wspomagający proces komunikacji między
studentami a pracownikami dziekanatu

Praca dyplomowa inżynierska

Napisana pod kierunkiem

dr inż. Romana Simińskiego

w roku akademickim 2024/2025

Katowice 2025

Spis treści

1	Wstęp	5
2	Analiza problemu.....	7
3	Analiza istniejących rozwiązań.....	13
3.1	Kryteria analizy porównawczej	13
3.2	Moduł podań w systemie USOS UŚ	13
3.3	System obiegu dokumentów EZD RP	15
3.4	Platforma Zendesk z licencją dla sektora edukacji.....	16
3.5	Platforma Slack dostosowana do środowiska edukacyjnego	17
3.6	Podsumowanie	19
4	Koncepcja własnego rozwiązania	21
4.1	Koncepcja rozwiązania użytkowego.....	21
4.2	Koncepcja rozwiązania technologicznego	22
5	Projekt ogólny.....	23
5.1	Specyfikacja wymagań funkcjonalnych i нефункциональных	23
5.1.1	Wymagania funkcjonalne.....	23
5.1.2	Wymagania нефункциональные	28
5.2	Architektura systemu.....	28
5.3	Metody i narzędzia realizacji	30
5.4	Koncepcja przechowywania danych	31
5.5	Projekt interfejsu użytkownika	33
6	Dokumentacja techniczna	39
6.1	Struktura warstwy logiki biznesowej.....	39
6.1.1	Główny pakiet systemu	40
6.1.2	Pakiet konfiguracyjny.....	41
6.1.3	Pakiet modelu danych i repozytoriów.....	42
6.1.4	Pakiet logiki biznesowej	43
6.1.5	Pakiet kontrolerów REST API.....	44
6.1.6	Pakiet komponentów bezpieczeństwa	47
6.1.7	Pakiet obsługi wyjątków.....	47
6.2	Struktura warstwy prezentacji.....	48
7	Testy i weryfikacja systemu	51
7.1	Testy jednostkowe.....	51
7.2	Testy integracyjne	59
7.3	Testy akceptacyjne	59
7.4	Testy responsywności	59

7.5	Podsumowanie procesu testowania	59
8	Przykładowy scenariusz wykorzystania systemu	60
8.1	Złożenie wniosku przez studenta	60
8.2	Komunikacja w ramach zgłoszenia	61
8.3	Zarządzanie zgłoszeniami przez administratora	61
9	Zakończenie	64
10	Bibliografia.....	66
11	Spis rysunków	67
12	Spis tabel	68

1 Wstęp

Popularyzacja usług teleinformatycznych przyniosła znaczne udogodnienia w sferze życia zawodowego, umożliwiając pracę zdalną oraz elektroniczne składanie wniosków. Niemniej jednak, pomimo rosnącej popularności tych innowacji, wiele instytucji nadal utrzymuje się w cyfrowej stagnacji. Doskonałym przykładem takiej opieszałości jest dziekanat studencki, który wciąż polega głównie na tradycyjnych dokumentach papierowych, dostarczanych osobiście do siedziby instytucji. W obliczu dzisiejszych możliwości realizacji większości spraw drogą internetową, taki sposób wymiany informacji, komunikatów, dokumentów i wniosków staje się nie tylko niewygodny, lecz również czasochłonny zarówno dla studentów, jak i personelu uczelni. Przy obecnej dostępności nowoczesnych narzędzi, takie podejście wydaje się anachroniczne i ogranicza efektywność działań administracyjnych.

Celem pracy jest zaprojektowanie i realizacja systemu wspomagania procesu komunikacji pomiędzy studentem a dziekanatem umożliwiającego wymianę informacji, wiadomości, wniosków w sposób bezkontaktowy. System ten umożliwi studentom składanie i odbieranie komunikatów, wniosków oraz dokumentów w formie elektronicznej poprzez zgłoszenia. Planuje się, że system umożliwi również efektywne zarządzanie dokumentami przez personel uczelni, z dostępem do nadesłanych, rozpatrzonych i zarchiwizowanych dokumentów. Dzięki odpowiedniemu grupowaniu i filtrowaniu dokumentów, proces rozpatrzenia wniosków stanie się bardziej efektywny. Zastosowanie tego systemu przyczyni się do skrócenia czasu całej procedury składania wniosków zarówno dla pracowników uczelni, jak i studentów. System ten zostanie zrealizowany w formie aplikacji internetowej, korzystając z nowoczesnych technologii, takich jak języki programowania Java i TypeScript, szablony Spring Boot, Angular, oraz silnik baz danych Microsoft SQL Server.

Rozdział drugi stanowi analizę głównego problemu, oferując zdefiniowanie jego kluczowych aspektów oraz ukazując kontekst i istotę wyzwania. W trzecim rozdziale dokonano pełnego przeglądu istniejących rozwiązań, łącząc go z krytycznym porównaniem do naszego projektowanego systemu. Koncepcje rozwiązania problemu, autorstwa inżyniera, zostały przedstawione w rozdziale czwartym, podczas gdy piąty rozdział poświęcony jest omówieniu ogólnej koncepcji organizacji systemu. W szóstym rozdziale odnajdziemy obszerną dokumentację techniczną projektu, zawierającą szczegółowe informacje na

temat zastosowanych technologii, narzędzi, struktury kodu, sposobów testowania oraz innych aspektów technicznych. Proces testowania i weryfikacji systemu pod kątem funkcjonalności został precyzyjnie opisany w siódmym rozdziale. Rozdział ósmy skupia się na prezentacji realnych scenariuszy i przypadków wykorzystania systemu, szczegółowo opisując kroki, interakcje i korzyści wynikające z implementacji opracowanego rozwiązania. Ostatni rozdział stanowi syntezę osiągnięć projektu, jednocześnie proponując perspektywy dalszego rozwoju.

2 Analiza problemu

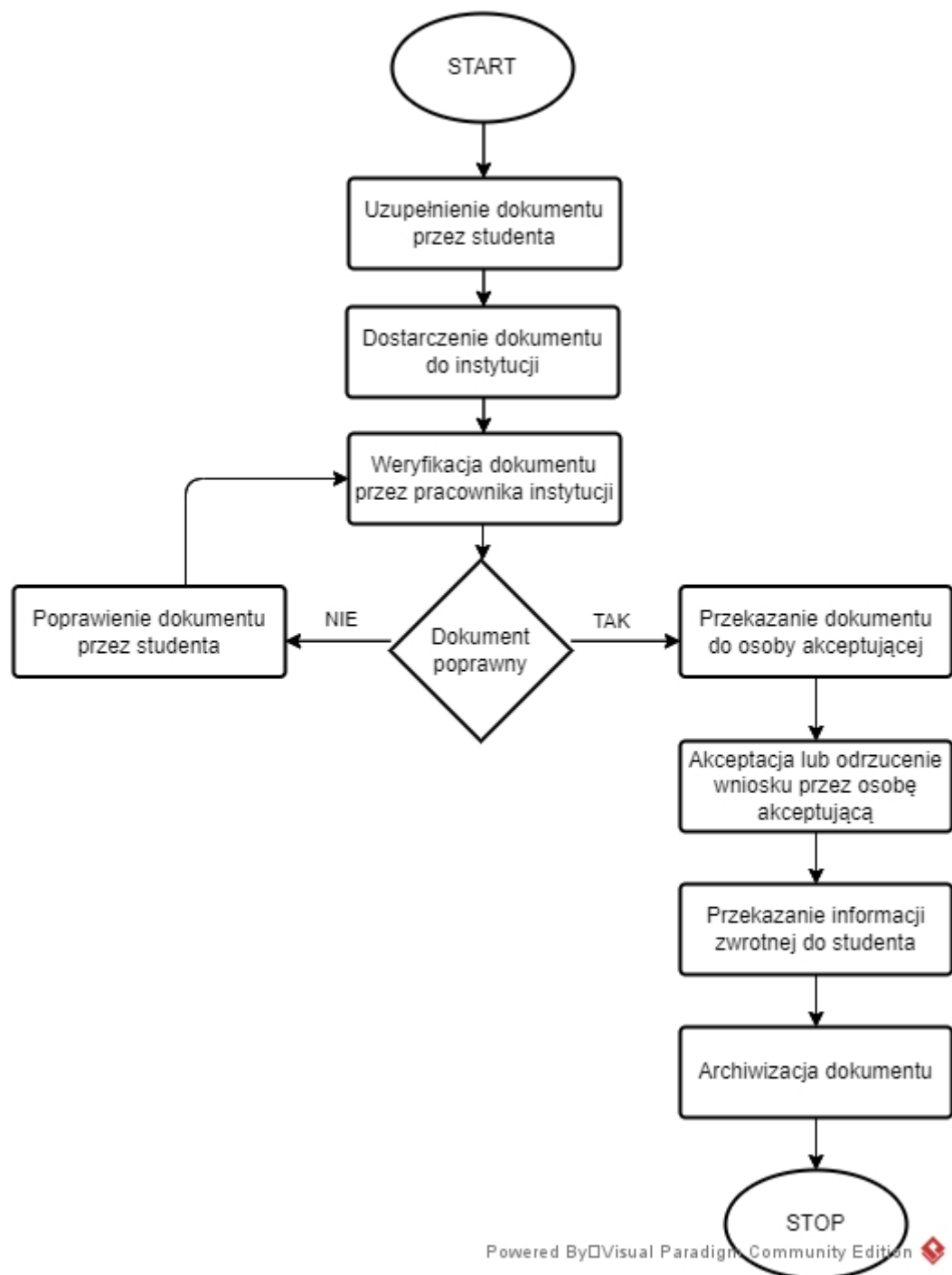
Podczas analizy potencjalnych problemów związanych z obecną formą wymiany informacji między studentami a pracownikami dziekanatu, autor zidentyfikował dwa potencjalne punkty inicjacji komunikacji. Student może zainicjować komunikację, gdy istnieje potrzeba uzyskania odpowiedzi na kwestie administracyjne, złożenia wniosku lub dostarczenia dokumentów. Założono, że jest to rodzaj komunikacji jeden do jednego, w której uczestniczą student i pracownik dziekanatu. Poza tym scenariuszem autor rozważył również możliwość, w której inicjatorem komunikacji będzie pracownik dziekanatu. Analizując ten przypadek, podzielono go na dwie potencjalne kategorie komunikacji. Pierwszą kategorią jest komunikacja jeden do jednego, podobnie jak w przypadku komunikacji inicjowanej przez studenta. Przykładem może być sytuacja, gdy od konkretnego studenta wymagane jest dostarczenie brakującego wniosku. Poza tym rodzajem komunikacji pojawić się może również komunikacja jeden do wielu, na przykład w formie rozsyłania informacji o konieczności wypełnienia obowiązkowych badań ankietowych przez określoną grupę studentów. Analizując obie możliwe ścieżki komunikacji - zarówno inicjowaną przez studenta, jak i pracownika dziekanatu - zauważamy wspólne problemy, ale także te, które występują tylko w jednym z tych przypadków.

W trakcie analizy problemów wynikających z komunikacji między studentem a pracownikiem dziekanatu, szczególnie zauważalnym aspektem jest przepływ dokumentów papierowych, który przedstawia Rysunek 1. Kwestia ta staje się problematyczna w sytuacjach, gdy od studenta wymaga się wypełnienia i dostarczenia określonych dokumentów w celu rozpoczęcia konkretnego procesu, na przykład wniosku o przeniesienie z formy studiów stacjonarnych na niestacjonarne, wzór wniosku przedstawia Rysunek 2. Proces ten nakłada szereg trudności na studenta, zaczynając od ograniczonego czasu, szczególnie dla tych, którzy mieszkają poza obszarem uczelni lub są zatrudnieni. Konieczność osobistego dostarczenia dokumentów może prowadzić do konieczności wzięcia urlopu lub przerwania innych zobowiązań, co stwarza dodatkowe utrudnienia. Ponadto, koszty podróży związane z koniecznością odwiedzenia dziekanatu mogą być istotnym obciążeniem dla studentów, zwłaszcza tych, którzy mieszkają daleko od uczelni. Dodatkowe wydatki związane z transportem publicznym, paliwem czy parkowaniem tylko potęgują trudności. To nie tylko sprawia trudności dla studentów, ale także generuje dodatkowe obciążenia dla

pracowników instytucji. Po dostarczeniu dokumentów, proces weryfikacji i udzielania odpowiedzi jest obecnie nieustrukturyzowany. Brak standaryzacji w przekazywaniu informacji zwrotnej, często opierającej się na rozmowach telefonicznych lub wiadomościach e-mail, może prowadzić do niejednoznaczności i kolejnych iteracji komunikacji między studentem a pracownikiem. Kolejnym aspektem jest problem archiwizacji i przechowywania dokumentów papierowych, co może wprowadzić nieefektywności zarówno w organizacji, jak i wydajności procesów. Zajmowanie przestrzeni fizycznej przez papierowe dokumenty oraz długotrwałe wyszukiwanie konkretnych dokumentów w archiwum stwarzają znaczące wyzwania. Podsumowując, istnieje potrzeba przemyślenia i ulepszenia procesu komunikacji oraz przepływu dokumentów, aby zminimalizować trudności zarówno dla studentów, jak i pracowników instytucji. Wprowadzenie bardziej efektywnych i zautomatyzowanych rozwiązań może znacząco poprawić tę sytuację.

Analiza zagadnień związanych z inicjacją komunikacji przez pracowników dziekanatu uwzględnia różne aspekty, szczególnie ilość odbiorców komunikatu w kontekście pojedynczego studenta lub grupy studentów. W przypadku komunikacji jednostronnej, gdzie pracownik instytucji pełni rolę nadawcy, a student jest odbiorcą, identyfikuje się szereg barier podobnych do tych obserwowanych w odwrotnym przepływie informacji, tj. od studenta do pracownika. Przykładowe trudności obejmują brak standardowego systemu przekazywania informacji dotyczących komunikatów, wydłużony czas niezbędny do archiwizacji oraz przeszukiwania dokumentów.

W kontekście omawianej ścieżki procesu pojawia się problem związanego z niepewnością pracownika co do faktycznego otrzymania i zapoznania się przez studenta z przekazany komunikatem. W przypadku indywidualnej korespondencji, gdzie pracownik instytucji pełni rolę nadawcy, a student jest odbiorcą, jedynym efektywnym środkiem eliminującym nieścisłości jest bezpośrednia rozmowa telefoniczna. W takim przypadku pracownik ma pewność, że przekazał informacje klarownie, a także ma możliwość udzielenia dodatkowych wyjaśnień oraz odpowiedzi na ewentualne pytania odbiorcy. Natomiast w sytuacji, gdy komunikacja odbywa się za pośrednictwem formy pisemnej, takiej jak wiadomość e-mail czy korespondencja, pracownik nie ma pewności, czy informacje zostały poprawnie przekazane, dopóki odbiorca nie udzieli informacji zwrotnej potwierdzającej zapoznanie się z treścią komunikatu.



Rysunek 1Aktualny obieg dokumentów w dziekanacie studenckim

..... (imię i nazwisko)	Katowice, dnia
..... (rok studiów – kierunek, stopień)	
..... (nr albumu)	
..... (nr telefonu)	
..... (e- mail)	
stacjonarne/niestacjonarne (forma studiów)	Dziekan Wydziału Nauk Ścisłych i Technicznych Uniwersytetu Śląskiego w Katowicach
<p>Zgodnie z § 18 ust. 3 Regulaminu studiów w Uniwersytecie Śląskim zwracam się z uprzejmą prośbą o wyrażenie zgody na przeniesienie ze studiów stacjonarnych/niestacjonarnych na studia niestacjonarne/stacjonarne.</p> <p>Prośbę swoją uzasadniam.....</p> <p>.....</p> <p>.....</p> <p>.....</p> <p>.....</p>	
 (podpis studenta)
Decyzja Dziekana:	
	Data wpływu pisma.....

Rysunek 2 Wniosek o przeniesienie z formy studiów stacjonarnych na niestacjonarne

Źródło: Przeniesienie (studia stacjonarne i niestacjonarne),
<https://us.edu.pl/wydzial/wnst/studia/student/regulaminy/>, 11.10.21

W kontekście omówionych wyzwań, planuje się wdrożenie systemu wspomagającego komunikację w postaci aplikacji internetowej. Wybór tego rozwiązania wynika z jego zdolności do obsługi zgłoszeń z dowolnego miejsca i o każdej porze dnia. Aplikacja internetowa stanowi efektywne narzędzie, które pozwoli studentom inicjować i monitorować procesy administracyjne bez konieczności fizycznego odwiedzania dziekanatu. Dzięki temu rozwiązaniu, ograniczenia czasowe związane z dostarczaniem dokumentów czy uzyskiwaniem odpowiedzi zostaną zredukowane, co wpłynie pozytywnie na wygodę studentów. Ponadto, aplikacja internetowa eliminuje konieczność ponoszenia kosztów związanych z podróżami do dziekanatu, co stanowi kolejny korzyść dla studentów. Dzięki dostępowi do zgłoszeń i informacji zwrotnych, proces komunikacji staje się bardziej przejrzysty i dostępny.

System ten umożliwi także pracownikom instytucji bardziej efektywne zarządzanie zgłoszeniami, co przyczyni się do usprawnienia procesów administracyjnych. Standaryzacja przekazywania informacji zwrotnych poprzez aplikację internetową pozwoli uniknąć niejednoznaczności i błędów wynikających z tradycyjnych form komunikacji.

Wprowadzenie aplikacji internetowej stanowi krok ku usprawnieniu całego procesu komunikacji między studentami a pracownikami dziekanatu, zwiększając jednocześnie dostępność i efektywność obsługi administracyjnej.

3 Analiza istniejących rozwiązań

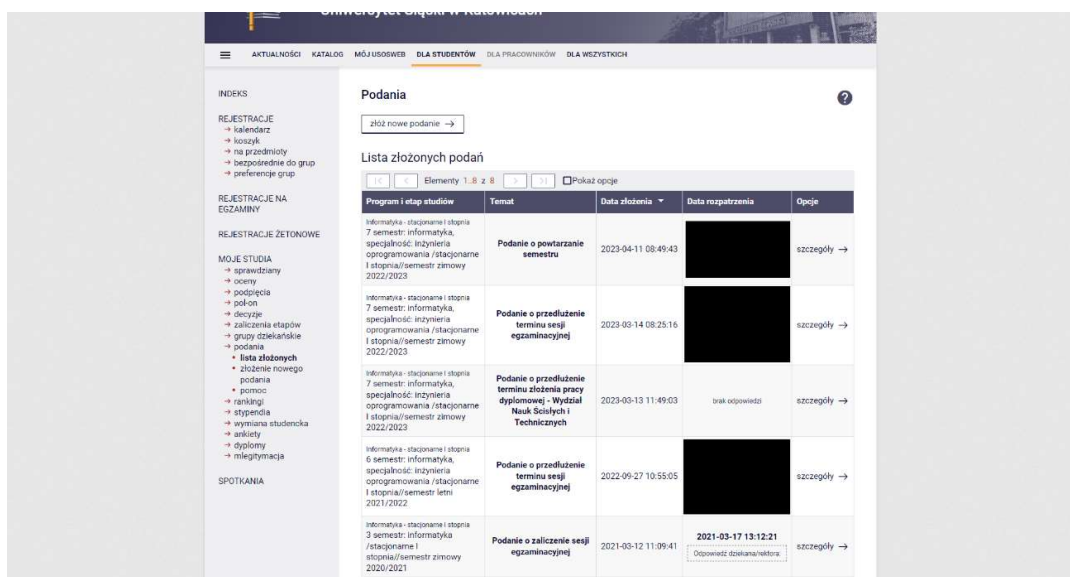
W rozdziale tym przedstawione zostaną rozwiązania, takie jak platformy, systemy czy moduły, które są już dostępne na rynku i mogą przyczynić się do rozwiązania problemów opisanych w rozdziale 2. Informacje na temat tych rozwiązań zostały pozyskane ze stron internetowych producentów lub dystrybutorów oprogramowania. Analiza będzie oparta na dostępnych danych dotyczących poszczególnych rozwiązań, umożliwiając pełniejsze zrozumienie ich potencjalnych korzyści i zastosowań.

3.1 Kryteria analizy porównawczej

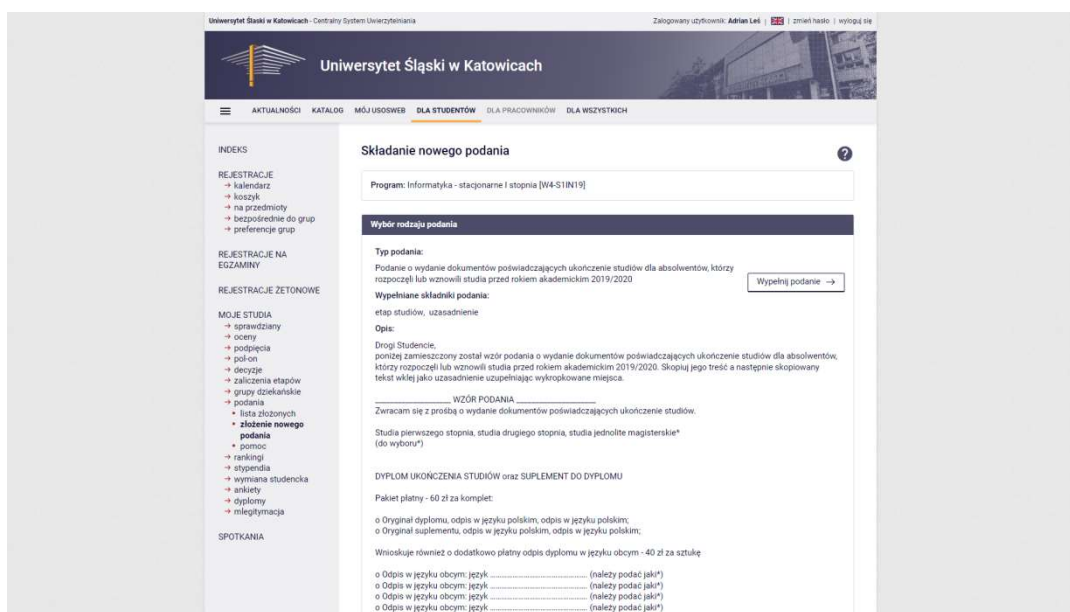
Analiza obejmować będzie następujące elementy, uwzględniające różnorodne kryteria w celu dokładnej oceny wad i zalet dostępnych rozwiązań. Kluczowe kryteria analizy porównawczej obejmują obecność dostępności funkcji komunikacyjnych, takich jak czaty i wiadomości prywatne. Ponadto, skupimy się na intuicyjności interfejsu użytkownika oraz ogólnej łatwości użycia, zwłaszcza dla różnych grup docelowych, takich jak studenci i pracownicy dziekanatu. Analizowane będą również możliwości związane z funkcjonalnością współpracy, w tym współdzieleniem plików i udzielaniem komentarzy. Wszystkie te kryteria umożliwią kompleksową ocenę potencjalnych wad i zalet każdego z rozwiązań.

3.2 Moduł podań w systemie USOS UŚ

Moduł podań stanowi integralną część systemu USOS UŚ, umożliwiając składanie określonych rodzajów podań w formie zdalnej. Składa się on z różnych elementów, z których głównym jest katalog, przedstawiony na rysunku 3, zawierający sprawy rozstrzygnięte lub aktualnie rozstrzygane, a także umożliwiające złożenie nowego podania. Proces składania nowego wniosku z katalogiem dostępnych dokumentów, jak przedstawiono na rysunku 4, staje się widoczny po naciśnięciu przycisku "Złóż nowe podanie" z okna głównego modułu. Po wyborze wniosku i naciśnięciu przycisku "Wypełnij podanie" użytkownik zostaje przeniesiony do okna wypełniania podania z instrukcją dotyczącą jego kompleksowego uzupełnienia.



Rysunek 3 Moduł podań systemu USOS UŚ



Rysunek 4 Składanie nowego podania w systemie USOS UŚ z listy określonych podań

Moduł podań w ramach systemu USOS UŚ nie obejmuje funkcji komunikacyjnych i współpracy; brak możliwości wymiany wiadomości prywatnych, uruchomienia czatu na określony temat, wymiany plików czy dodania komentarzy. Moduł ten skupia się jedynie na umożliwieniu wypełnienia podania i prezentuje ostateczną odpowiedź na zgłoszenie. Mimo braku zaawansowanych funkcji komunikacyjnych, moduł charakteryzuje się spójnym

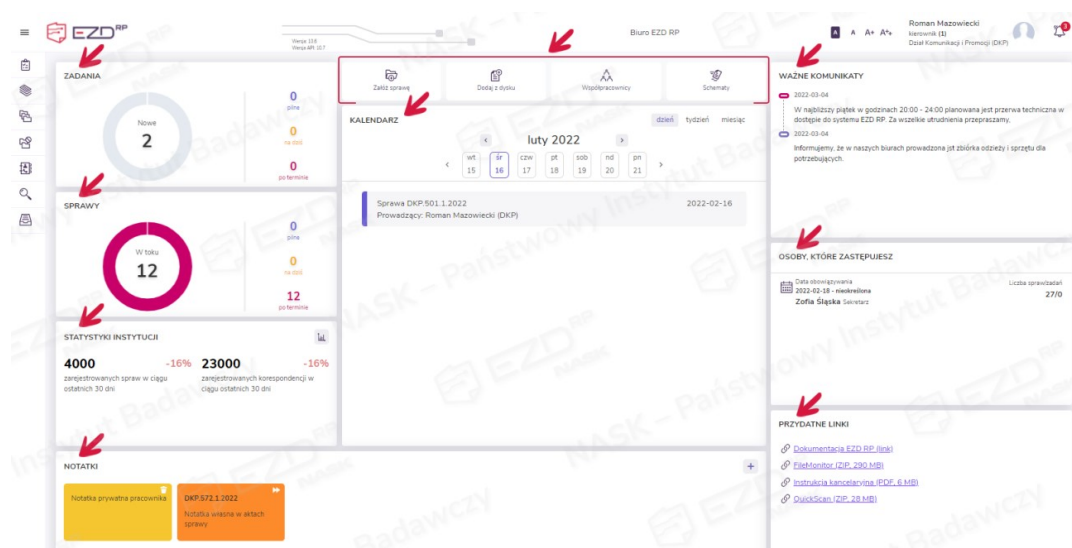
interfejsem, klarowną nawigacją i czytelnym przedstawianiem informacji. Jego głównym celem jest efektywne umożliwienie składania podań i prezentacja końcowego stanu odpowiedzi.

3.3 System obiegu dokumentów EZD RP

System elektronicznego zarządzania dokumentacją EZD RP umożliwia szereg funkcji, w tym prowadzenie korespondencji w formie elektronicznej, wspierając tym samym efektywną komunikację wewnętrzną jednostki. Zgodnie z dostępnym opisem na stronie producenta, system ten został specjalnie opracowany, aby sprostać potrzebom biznesowym polskiej administracji publicznej. EZD RP nie tylko umożliwia elektroniczne rozstrzyganie spraw, ale również wspiera obsługę spraw w formie papierowej. Oprogramowanie odzwierciedla kluczowe procesy wykonywane przez urzędników podczas załatwiania spraw. EZD RP wyróżnia się także wspieraniem efektywnego obiegu dokumentów za pomocą systemu zadań. Dzięki tej funkcji, czas obiegu dokumentów jest redukowany, co z kolei przekłada się na zmniejszenie liczby koniecznej korespondencji. To wszystko wspólnie sprawia, że EZD RP stanowi kompleksowe narzędzie wspomagające zarządzanie dokumentacją oraz procesami administracyjnymi w kontekście polskiej administracji publicznej¹.

EZD RP zapewnia możliwość prowadzenia korespondencji w formie elektronicznej, w pełni spełniając wymagania związane z funkcjami komunikacyjnymi. System doskonale wspiera obieg elektronicznych dokumentów, umożliwiając dostęp do nich dla osób uczestniczących w procesie obiegu. W ten sposób efektywnie spełnia również wymagania dotyczące współpracy nad dokumentami. Jednolity i intuicyjny interfejs systemu, przedstawiony na rysunku 5, znacznie ułatwia korzystanie z jego funkcji, co przyczynia się do wygodnego i efektywnego zarządzania dokumentacją elektroniczną.

¹ O systemie, <https://www.gov.pl/web/ezd-rp/o-ezd-rp>



Rysunek 5 Ekran startowy systemu ERP RP

źródło: Podręcznik użytkownika systemu EKD RP, podstawy użytkowania, ekran startowy, <https://podrecznik.ezdrp.gov.pl/ekran-startowy/>, 22.08.2023

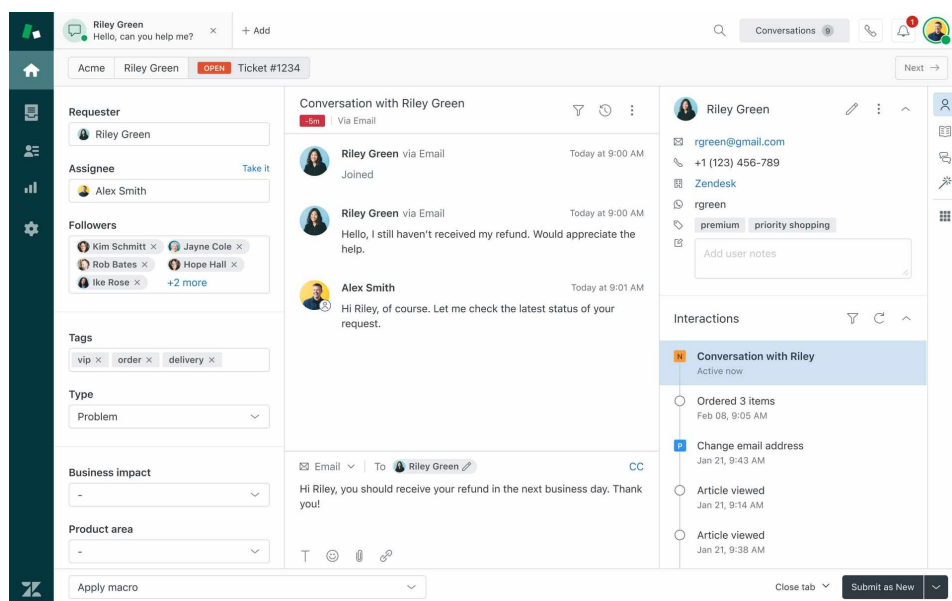
3.4 Platforma Zendesk z licencją dla sektora edukacji

Platforma Zendesk oferuje dedykowane licencje dostosowane specjalnie do sektora edukacyjnego, dostarczając kompleksowe narzędzia do efektywnego zarządzania w instytucjach edukacyjnych. Dzięki temu specjalnemu rozwiązaniu, szkoły, uczelnie i inne instytucje edukacyjne mogą skorzystać z zaawansowanych funkcji Zendesk, dostosowanych do ich unikalnych potrzeb. Zendesk wspiera efektywną komunikację między pracownikami dziekanatu, a studentami poprzez system biletów, czaty online i możliwość śledzenia zgłoszeń. Dodatkowo, platforma umożliwia zautomatyzowane przypisywanie i rozwiązywanie zadań, co skutkuje szybszą reakcją na pytania oraz problemy studentów².

Możliwość dostosowania do specyfiki instytucji czyni Zendesk dedykowanym rozwiązaniem wspierającym efektywną komunikację w sektorze edukacyjnym. Platforma oferuje funkcję dodawania załączników do biletów, co umożliwia przysyłanie plików w ramach komunikacji między pracownikami dziekanatu, a studentami. Interfejs Zendesk został zaprojektowany w sposób umożliwiający łatwe nawigowanie i korzystanie z różnych funkcji platformy. Dzięki temu

² Zendesk for Education, <https://www.zendesk.com/education/>

użytkownicy mogą efektywnie korzystać z dostępnych narzędzi, co przyczynia się do płynności i skuteczności procesu komunikacji.



Rysunek 6 Ekran główny platformy Zendesk

źródło: <https://www.zendesk.com/register/>

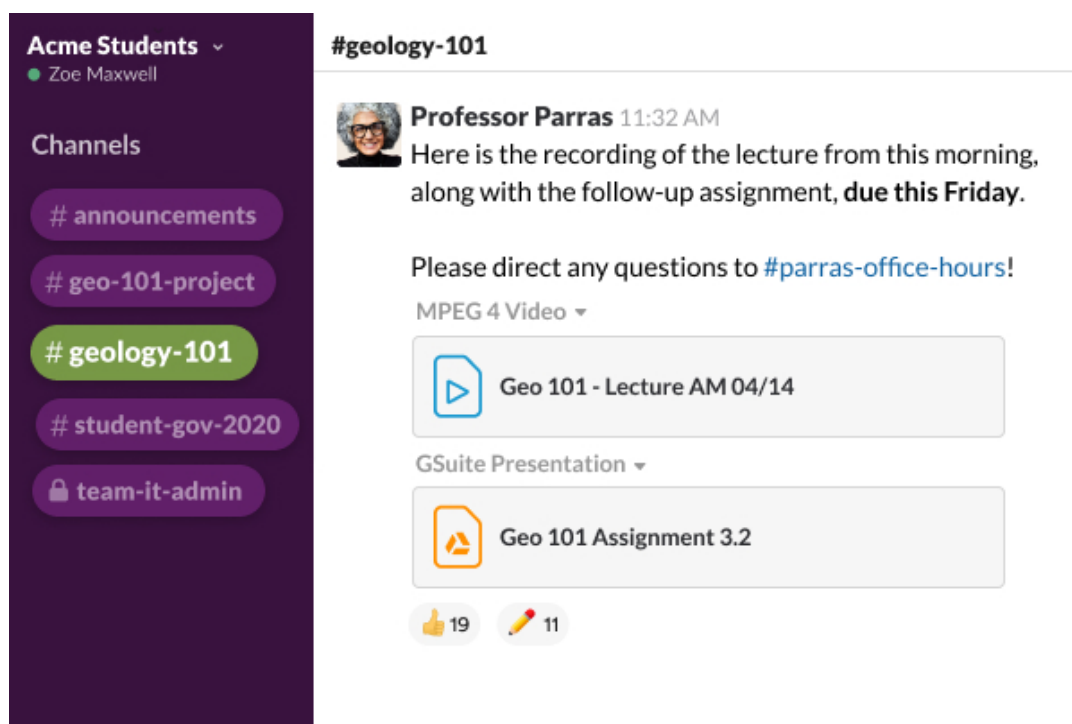
3.5 Platforma Slack dostosowana do środowiska edukacyjnego

Platforma Slack, dostosowana do kontekstu edukacyjnego, pełni znaczącą rolę jako skuteczne narzędzie ułatwiające komunikację między pracownikami uczelni a studentami. Wykorzystanie kanałów tematycznych pozwala na precyzyjne uporządkowanie dyskusji związanych z różnymi przedmiotami, projektami badawczymi czy wydarzeniami kampusowymi. Pracownicy dziekanatu korzystają z platformy do szybkiego przekazywania informacji dotyczących aktualnych wydarzeń na kampusie, udostępniania materiałów akademickich oraz przekazywania ważnych ogłoszeń³.

Dzięki funkcji czatu w czasie rzeczywistym, studenci mogą aktywnie uczestniczyć w procesie komunikacji, zadając pytania i otrzymując natychmiastowe odpowiedzi od pracowników uczelni. Ta interaktywna forma komunikacji istotnie przyczynia się do poprawy jakości dialogu, umożliwiając

³ Your guide to Slack for higher education, <https://slack.com/resources/using-slack/your-guide-to-slack-for-higher-education>

szybka i skuteczną wymianę informacji. Dodatkowo, możliwość współdzielenia plików w ramach platformy Slack ułatwia przesyłanie dokumentów, co sprzyja efektywnej wymianie informacji. Te rozbudowane funkcje sprawiają, że Slack, dostosowany do środowiska edukacyjnego, staje się platformą sprzyjającą efektywnej i dynamicznej komunikacji między pracownikami uczelni a studentami. Interfejs Slacka został zaprojektowany w taki sposób, aby umożliwić użytkownikom łatwe poruszanie się po platformie i korzystanie z różnych funkcji bez zbędnych trudności. To z kolei przyczynia się do płynnego i intuicyjnego doświadczenia użytkownika, co jest istotne dla efektywnego wykorzystania potencjału komunikacyjnego platformy w kontekście edukacyjnym.



Rysunek 7 Interfejs platformy Slack

źródło: <https://slack.com/solutions/distance-learning>

3.6 Podsumowanie

Podsumowując analizę porównawczą różnych rozwiązań, można zauważyć, że każde z badanych modułów, systemów czy platform posiada swoje unikalne zalety oraz oferuje różnorodne funkcje, które mogą być dostosowane do specyfiki działania dziekanatu.

W przypadku modułu podań systemu USOS UŚ, pomimo braku pewnych funkcjonalności związanych z komunikacją i współpracą, wartością dodaną jest spójny i intuicyjny interfejs oraz klarowna nawigacja, co ułatwia obsługę systemu.

System elektronicznego zarządzania dokumentacją EZD RP w pełni spełnia przyjęte kryteria funkcjonalne, umożliwiając efektywną komunikację i wspierając przejrzystość procesów dzięki możliwości dołączania załączników i komentowania.

Platforma Zendesk, dzięki dedykowanej licencji edukacyjnej, dostosowuje się do specyfiki instytucji, efektywnie wspierając komunikację i wymianę informacji. System biletowy, możliwość dołączania załączników oraz intuicyjny interfejs sprawiają, że jest kompleksowym narzędziem w obszarze obsługi klienta.

Platforma Slack, dostosowana do środowiska edukacyjnego, wyróżnia się funkcją czatu w czasie rzeczywistym, umożliwiając aktywną komunikację. Możliwość współdzielenia plików oraz przyjazny interfejs stanowią dodatkowe atuty wspierające efektywną komunikację wśród użytkowników.

4 Koncepcja własnego rozwiązania

W rozdziale tym przedstawiona zostanie koncepcja innowacyjnego systemu wspierającego wymianę wiadomości między studentem a dziekanatem, opartego na zaawansowanym systemie biletowym. Zaproponowane rozwiązanie skoncentruje się wyłącznie na efektywnej komunikacji, eliminując funkcje administracyjne. System oparty na biletach umożliwi studentom zgłaszanie różnorodnych spraw, takich jak pytania dotyczące przedmiotów, prośby o informacje czy sytuacje wymagające wsparcia ze strony dziekanatu. Każde zgłoszenie zostanie przypisane do unikatowego numeru biletu, co ułatwi śledzenie postępu i zapewni transparentność procesu. Prostota funkcji systemu ma na celu skoncentrowanie się na usprawnieniu komunikacji, bez zbędnych elementów administracyjnych. Interfejs użytkownika zostanie zaprojektowany w sposób przyjazny i intuicyjny, aby ułatwić studentom korzystanie z systemu, a tym samym zminimalizować potencjalne bariery w komunikacji z dziekanatem. Wprowadzenie tego systemu ma na celu efektywne zarządzanie wymianą informacji, poprawę komunikacji oraz zwiększenie satysfakcji studentów z obsługi studenckiej.

4.1 Koncepcja rozwiązania użytkowego

W ramach proponowanej koncepcji praktycznego systemu wymiany informacji między studentami a dziekanatem, opartego na systemie biletowym, priorytetem jest zapewnienie prostoty obsługi i szybkości reakcji. Interfejs tego narzędzia zostanie starannie zaprojektowany, aby każdy student mógł łatwo zrozumieć, jak korzystać z systemu, nie wymagając specjalistycznej wiedzy technicznej. Aby zgłosić sprawę, studenci będą mieli dostęp do intuicyjnego formularza, który umożliwi im precyzyjne opisanie problemu. Generowanie unikalnych numerów biletów pozwoli na szybkie i precyzyjne identyfikowanie zgłoszeń, a także umożliwi studentom śledzenie postępu w ich rozpatrywaniu.

System ten uwzględni również możliwość natychmiastowych powiadomień, informujących studentów o statusie ich zgłoszeń oraz ewentualnych dodatkowych krokach do podjęcia. Dziekanat będzie mógł efektywnie zarządzać zgłoszeniami, przypisując odpowiednich pracowników do ich rozpatrzenia. Dodatkowo, wbudowane funkcje umożliwią dołączanie załączników i komentarzy do zgłoszeń.

Wprowadzenie tego systemu ma na celu nie tylko zwiększenie efektywności komunikacji między studentami a dziekanatem, ale także poprawę jakości obsługi studenckiej, redukcję czasu oczekiwania na odpowiedzi oraz zwiększenie satysfakcji studentów z procesów administracyjnych na uczelni. To narzędzie nie tylko ułatwi wymianę informacji, ale również przyczyni się do bardziej efektywnego i zintegrowanego zarządzania sprawami studenckimi. Dzięki takiemu podejściu, procesy administracyjne staną się bardziej dostępne, zrozumiałe i efektywne dla wszystkich zainteresowanych strony.

4.2 Koncepcja rozwiązania technologicznego

Koncepcja rozwiązania technologicznego systemu wymiany wiadomości pomiędzy studentem a pracownikami dziekanatu oparta jest na architekturze REST, stanowiącej kluczowy element projektu. Aplikacja internetowa zostanie opracowana przy użyciu języków Java i TypeScript, wykorzystując szablony Spring i Angular oraz silnika bazy danych Microsoft SQL Server. Architektura REST, umożliwi spójną i efektywną komunikację pomiędzy klientem a serwerem, co jest kluczowe dla dynamicznego obiegu informacji w systemie. W kontekście operacji CRUD dla zgłoszeń, REST pozwoli na zaimplementowanie prostych i jednolitych interfejsów, umożliwiając studentom zgłaszanie, przeglądanie, aktualizację i usuwanie swoich zgłoszeń. To z kolei ułatwi śledzenie postępu w rozpatrywaniu spraw, zgodnie z zasadami architektury REST. Do transportu danych pomiędzy klientem a REST API użyty zostanie format JSON. Dodatkowo, w ramach tej koncepcji, wykorzystany zostanie silnik bazy danych Microsoft SQL Server w celu skutecznego przechowywania i zarządzania danymi systemowymi. Microsoft SQL Server oferuje nie tylko wysoką wydajność, ale także solidne mechanizmy zabezpieczające integralność danych. Dzięki temu, informacje dotyczące zgłoszeń, komunikatów i historii interakcji między studentami a pracownikami dziekanatu będą przechowywane w sposób niezawodny i dostępne dla systemu w czasie rzeczywistym. Implementacja architektury REST w połączeniu z silnikiem bazy danych MSSQL ma na celu zapewnienie skalowalności, elastyczności oraz wysokiej wydajności systemu. To kompleksowe podejście pozwala skoncentrować się na efektywnej wymianie informacji między użytkownikami, jednocześnie gwarantując solidne zarządzanie danymi oraz zgodność z nowoczesnymi standardami technologicznymi.

5 Projekt ogólny

Rozdział ten prezentuje ogólny projekt systemu wspomagającego komunikację między studentami a pracownikami dziekanatu, skupiający się na efektywnym przepływie informacji i optymalizacji procesów komunikacyjnych.

5.1 Specyfikacja wymagań funkcjonalnych i нефункциональных⁴

5.1.1 Wymagania funkcjonalne

W sekcji dotyczącej wymagań funkcjonalnych przedstawione są szczegółowe opisy funkcji i operacji, które powinny być dostępne w systemie. Te wymagania obejmują konkretny zakres działań, jakie użytkownicy i system powinni być w stanie wykonać, a także opisują oczekiwane rezultaty tych operacji.

Tabela 1 Wymagania funkcjonalne

Nazwa wymagania	Opis wymagania
Logowanie	System musi umożliwiać użytkownikowi zalogować się na własne konto za pomocą nazwy użytkownika i hasła.
Wylogowanie	System musi umożliwić użytkownikowi zakończenie sesji poprzez odebranie dostępu do aplikacji dla osób, które nie są zalogowane.
Zmiana hasła	System powinien umożliwiać użytkownikowi dowolną zmianę hasła dostępowego do swojego konta w trakcie korzystania z systemu. Niemniej jednak, zmiana hasła powinna być obowiązkowa przy pierwszym logowaniu do systemu.
Blokada użytkownika	System musi umożliwiać administratorowi zablokowanie dostępu użytkownika do zasobów systemu.

⁴ Rozdział 5. Specyfikacja wymagań, „Specyfikacja wymagań oprogramowania. Kluczowe praktyki analizy biznesowej”, K. Wiegers, C. Hokanson, 2024, str. 112

Odblokowanie użytkownika	System musi umożliwić administratorowi odblokowanie dostępu użytkownika do zasobów systemu.
Dodanie użytkownika	System musi umożliwić administratorowi systemu dodanie nowego użytkownika poprzez wprowadzenie danych osobowych. Następnie, na podstawie tych informacji, system powinien generować dane logowania i przysyłać je na adres e-mail nowego użytkownika.
Archiwizacja użytkownika	System musi umożliwić archiwizację danych użytkowników, jednocześnie blokując dostęp do systemu dla danego użytkownika. Ponadto, system powinien archiwizować zgłoszenia wprowadzone przez tego użytkownika. Administratorowi oraz użytkownikowi nadrzędnemu powinno być udostępnione uprawnienie do przeglądania zarchiwizowanych danych.
Przegląd użytkowników	System musi umożliwić administratorowi przegląd listy użytkowników uprawnionych do korzystania z systemu. Ta lista powinna zawierać informacje takie jak nazwa użytkownika oraz daty i czasy ostatnich logowań użytkowników.
Utworzenie zgłoszenia	System musi umożliwić użytkownikowi utworzenie nowego zgłoszenia poprzez wybór odpowiedniej kategorii, opisanie problemu, wprowadzenie ewentualnego komentarza oraz dodanie załącznika do zgłoszenia. Podczas tworzenia zgłoszenia, system generuje automatycznie unikalną etykietę identyfikacyjną, która służy do jednoznacznego identyfikowania danego zgłoszenia.
Zamknięcie zgłoszenia	System musi umożliwić użytkownikowi zamknięcie zgłoszenia przed jego zakończeniem, przy czym użytkownik powinien być w stanie podać przyczynę zamknięcia zgłoszenia.

Przegląd zgłoszeń	System musi umożliwić użytkownikowi dostęp do listy aktywnych zgłoszeń utworzonych przez niego. Administrator systemu oraz osoba nadrzędna powinny mieć pełny wgląd we wszystkie zgłoszenia, niezależnie od tego, kto je utworzył.
Historia zgłoszeń	System musi umożliwić użytkownikowi wgląd w historię zamkniętych zgłoszeń, które zostały utworzone przez niego od momentu rozpoczęcia korzystania z systemu. Jednocześnie, zarówno administratorzy, jak i użytkownicy nadrzędni, powinni posiadać pełny dostęp do wszystkich zgłoszeń, zarówno aktywnych, jak i zamkniętych.
Monitorowanie statusu zgłoszenia	System musi umożliwić użytkownikowi monitorowanie aktualnego statusu zgłoszenia, umożliwiając dostęp do informacji na temat bieżącego postępu w rozpatrywaniu zgłoszenia oraz ewentualnych zmian w jego statusie.
Archiwizacja zgłoszeń	System musi umożliwić archiwizację zgłoszeń, które zostały wprowadzone przez użytkowników. Proces archiwizacji powinien obejmować wszystkie istotne informacje, takie jak załączniki i komentarze zawarte w zgłoszeniach. Dzięki temu użytkownicy i administratorzy będą mieli dostęp do pełnej historii zgłoszeń, włączając w to wszelkie istotne szczegóły, nawet po ich zamknięciu.
Kategoryzowanie zgłoszeń	System musi umożliwić podział zgłoszeń na kategorie w zależności od określonych kryteriów bądź parametrów. Ten podział na kategorie ułatwi organizację zgłoszeń oraz pozwoli użytkownikom i administratorom systemu skoncentrować się na konkretnych rodzajach problemów lub zagadnień.

Filtrowanie zgłoszeń	System musi umożliwić filtrowanie listy zgłoszeń na podstawie danych zawartych w polach zgłoszenia, informacji o użytkowniku oraz aktualnym wykonawcy. Dzięki temu użytkownicy i administratorzy będą w stanie szybko odnaleźć konkretne zgłoszenia w zależności od określonych kryteriów, co poprawi efektywność zarządzania zgłoszeniami.
Priorytetyzacji zgłoszeń	System musi posiadać możliwość ustalanie priorytetów dla różnych kategorii zgłoszeń. Dzięki temu, administratorzy mogą zdefiniować, które kategorie są bardziej priorytetowe niż inne, co ułatwi skoncentrowanie się na rozwiązywaniu najważniejszych problemów w pierwszej kolejności. Priorytety te mogą być dostosowywane zgodnie z aktualnymi potrzebami i sytuacją w systemie.
Sortowanie zgłoszeń	System musi umożliwić sortowanie listy zgłoszeń, umożliwiając użytkownikom wybór kategorii, według których chcieliby posortować zgłoszenia. To obejmuje możliwość sortowania po różnych danych zawartych w polach zgłoszeń, na przykład według daty utworzenia, priorytetu, kategorii czy aktualnego statusu. Dzięki temu użytkownicy będą mogli dostosować widok listy zgłoszeń do swoich indywidualnych potrzeb i priorytetów.
Komentarze w zgłoszeniach	System musi umożliwić dodawanie komentarzy do zarówno aktywnych, jak i zamkniętych zgłoszeń. To umożliwi użytkownikom oraz administratorom systemu udzielanie dodatkowych informacji, dzielenie się uwagami lub dostarczanie dodatkowego kontekstu dotyczącego zgłoszenia. Komentarze te mogą być ważne zarówno podczas trwania procesu rozpatrywania zgłoszenia, jak i w jego historii po zamknięciu.

Załączniki w zgłoszeniach	System musi umożliwić dołączanie załączników do zgłoszeń, przy czym akceptowane formaty plików dla załączników to PDF, JPG i PNG. To pozwoli użytkownikom oraz administratorom systemu dołączać istotne dokumenty, obrazy czy inne pliki do zgłoszeń, co może być istotne dla kompleksowej analizy i rozwiązania problemu zgłaszanego w zgłoszeniu.
Wyszukiwanie zgłoszeń	System musi umożliwić użytkownikowi wyszukiwanie zgłoszeń na podstawie wygenerowanej unikatowej etykiety. Dzięki temu użytkownik będzie w stanie szybko odnaleźć konkretne zgłoszenie, używając unikalnego identyfikatora przypisanego do danego zgłoszenia. To usprawni proces zarządzania zgłoszeniami i umożliwi skuteczne śledzenie ich statusu.
Powiadomienia e-mail	System musi umożliwić wysyłanie powiadomień e-mail do użytkowników. Te powiadomienia e-mailowe mogą zawierać informacje na temat danych logowania, nowych zgłoszeń, zmianie statusu zgłoszenia, nowych komentarzach oraz załącznikach. Dzięki temu użytkownicy będą bieżąco informowani o istotnych wydarzeniach związanych z systemem, co przyczyni się do szybszej reakcji na zmiany oraz skuteczniejszej komunikacji.
Powiadomienia typu push	System musi posiadać możliwość wysyłania powiadomień push do użytkowników wewnątrz aplikacji. Te powiadomienia push mogą przekazywać informacje dotyczące danych logowania, nowych zgłoszeń, zmiany statusu zgłoszenia, nowych komentarzy oraz załączników. Dzięki temu użytkownicy będą natychmiastowo informowani o istotnych zdarzeniach bez konieczności aktywnego sprawdzania aplikacji, co przyczyni się do zwiększenia efektywności i bieżącej komunikacji w systemie.

5.1.2 Wymagania niefunkcjonalne

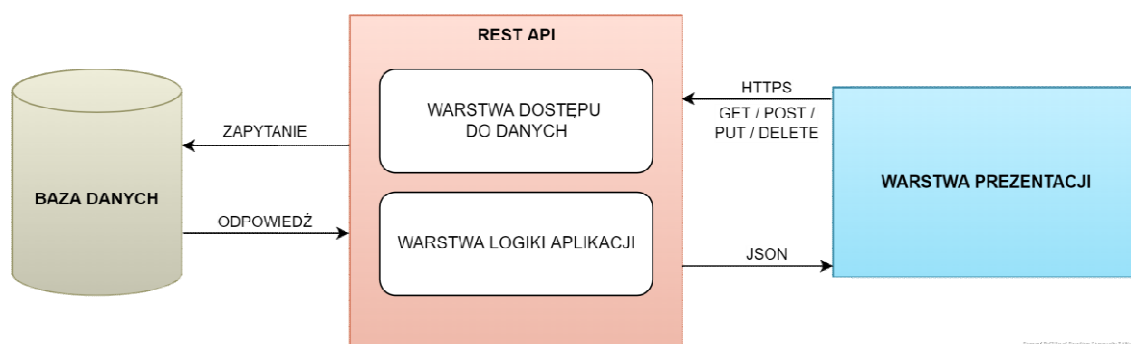
W zakresie wymagań niefunkcjonalnych określa się ograniczenia dotyczące wydajności, bezpieczeństwa, dostępności oraz innych kluczowych aspektów systemu. Te wymagania koncentrują się na parametrach, które nie mają bezpośredniego związku z możliwościami dostępnymi w systemie, lecz odgrywają kluczową rolę w efektywności, stabilności i bezpieczeństwie systemu. Obejmują one kwestie takie jak czas odpowiedzi, poziom bezpieczeństwa danych, dostępność systemu oraz inne istotne cechy, które mają wpływ na ogólną jakość obsługi systemu.

1. System powinien zostać zaprojektowany w architekturze klient – serwer.
2. Dostępność systemu przez całą dobę, siedem dni w tygodniu, z minimalnymi okresami niedostępności związanych z koniecznymi pracami konserwacyjnymi.
3. System powinien jednocześnie obsługiwać co najmniej 500 studentów, utrzymując wysoką wydajność.
4. Podczas komunikacji pomiędzy klientem a serwerem zastosowane powinno zostać szyfrowanie danych, aby skutecznie zabezpieczyć prywatność przesyłanych informacji.
5. System powinien zawierać autoryzację i uwierzytelnianie, w celu zagwarantowania dostępu jedynie uprawnionym osobom.
6. Interfejs użytkownika powinien zostać zaprojektowany w sposób intuicyjny i łatwy w obsłudze, co przyczyni się do wygodnego korzystania z systemu.
7. System powinna cechować spójna kolorystyka w charakteryzacji obiektów, co przyczyni się do jednolitego i estetycznego wyglądu interfejsu.

5.2 Architektura systemu

System wspomagający procesy komunikacji między studentem a dziekanatem zostanie zrealizowany jako aplikacja internetowa, oparta na architekturze klient-serwer. Projekt obejmie trójwarstwową strukturę. Dane wykorzystywane przez aplikację będą przechowywane w bazie danych utworzonej na dedykowanym serwerze, a dostęp do nich będzie zapewniała warstwa dostępu do danych. Warstwa logiki biznesowej, odpowiedzialna za obsługę logiki

aplikacji, wraz z warstwą dostępu do danych, utworzy interfejs programistyczny (API) odpowiedzialny za komunikację między warstwą kliencką a bazą danych. Warstwa prezentacji, działająca w środowisku przeglądarki, będzie komunikować się z warstwą logiki biznesowej za pośrednictwem punktów końcowych REST API. Taka struktura warstwowa pozwoli na efektywną organizację procesów komunikacji i zapewni optymalną wydajność systemu.



Rysunek 8 Schemat architektury REST API

źródło: <https://spring.academy/courses/building-a-rest-api-with-spring-boot/lessons/implementing-get>

Warstwa prezentacji odpowiedzialna jest za interakcje z użytkownikiem i prezentację informacji. Odpowiada ona za zbieranie danych od użytkowników poprzez na przykład formularze, prezentację informacji oraz interakcję z warstwą logiki biznesowej na podstawie akcji użytkownika. Warstwa logiki biznesowej zawiera funkcje i reguły biznesowe systemu. Odpowiada za przetwarzanie poleceń otrzymanych z warstwy prezentacji, realizując je i zwracając odpowiedź na zapytanie. Kontroluje przepływ danych i decyzje w systemie. Zarządza operacjami, a także implementuje reguły biznesowe i logikę operacyjną. Warstwa dostępu do danych odpowiada za komunikację z bazą danych i manipulację danymi. Realizuje operacje odczytu, zapisu, aktualizacji i usuwania danych w bazie danych. Zapewnia integralność danych i efektywnie zarządza nimi w kontekście systemu.⁵Opisany proces komunikacji między warstwami systemu opiera się na schemacie żądań i odpowiedzi, który jest przetwarzany za pomocą REST API. Inicjatywa zaczyna się w warstwie prezentacji, gdzie generowane jest żądanie HTTP. Następnie to żądanie

⁵ Architectural design, "Software Engineering: A Practitioner's Approach.", Pressman, R. S., McGraw-Hill Education, 2014, s. 242 - 275

przekazywane jest do warstwy logiki biznesowej, która podejmuje decyzję odnośnie do operacji, jakie powinna wykonać, komunikując się jednocześnie z warstwą dostępu do danych i bazą danych. Po otrzymaniu odpowiedzi z bazy danych, warstwa dostępu do danych przekazuje te informacje z powrotem do warstwy logiki biznesowej. W tej warstwie dane są dalej przetwarzane, a następnie w formie odpowiedzi przekazywane z powrotem do warstwy prezentacji. Cały ten proces ma na celu zapewnienie efektywnej komunikacji i współpracy pomiędzy poszczególnymi warstwami systemu.

5.3 Metody i narzędzia realizacji

System wspierający wymianę informacji pomiędzy studentem a dziekanatem, oparty na architekturze REST API, zostanie zbudowany przy użyciu nowoczesnych narzędzi, gwarantujących stabilność aplikacji. Dane używane przez aplikację będą przechowywane w relacyjnej bazie danych opartej na silniku Microsoft SQL Server 2019 (wersja 15.0.2000.5), cechującej się wysoką wydajnością i solidnymi mechanizmami zabezpieczającymi integralność danych.

Część serwerowa systemu zostanie zrealizowana przy użyciu szablonu Spring Boot, umożliwiającego łatwe tworzenie RESTful API za pomocą języka Java. Spring Boot automatyzuje konfigurację platformy Spring oraz bibliotek Javy, co znacznie upraszcza proces konfiguracji systemu.⁶ W projekcie zostanie używana wersja języka Java 17.0.9, a szablon Spring Boot w wersji 3.2.2. Do zarządzania bibliotekami i automatyzacji budowy projektu posłuży narzędzie Gradle w wersji 7.5. Dane przekazywane w odpowiedzi na zapytania klienta przez serwer będą w standardzie JSON.

Warstwa prezentacji zostanie stworzona przy użyciu platformy Angular w wersji 19.2.0, opartej na języku TypeScript. Angular umożliwia tworzenie aplikacji jednostronicowych, sterowanych poprzez komponenty i elementy nawigacyjne, co pozwala na płynne przełączanie się między widokami.⁷ Do kontroli nad wyglądem aplikacji użyta zostanie biblioteka ng-bootstrap (wersja 18.0.0), umożliwiająca korzystanie z elementów szablonu Bootstrap do efektywnego projektowania interfejsu użytkownika.

⁶ Spring Boot, <https://spring.io/projects/spring-boot>

⁷ Using Angular routes in a single-page application, <https://angular.io/guide/router-tutorial>

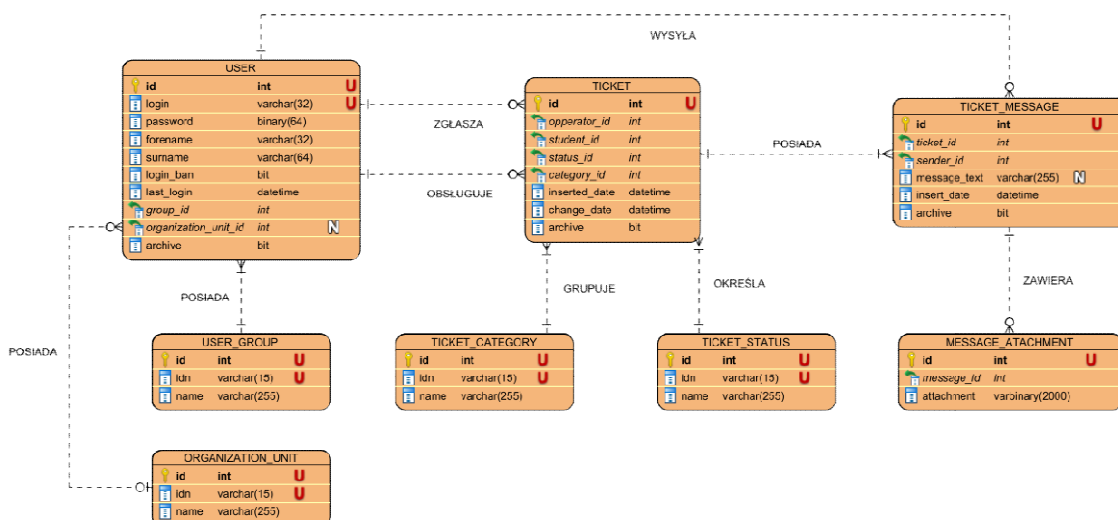
5.4 Koncepcja przechowywania danych

System wspomagania komunikacji wykorzysta relacyjny model bazy danych, którego tworzenie będzie oparte na silniku Microsoft SQL Server 2022. Wybór tego rozwiązania niesie ze sobą wiele korzyści. Dzięki zaawansowanym funkcjom zabezpieczeń oferowanym przez Microsoft SQL Server, takim jak autoryzacja i kontrola dostępu, będziemy mieć pewność co do bezpiecznego przechowywania informacji. Dodatkowo, silnik SQL Server wyposażony jest w mechanizmy optymalizacji, takie jak indeksowanie, co umożliwia szybkie przetwarzanie danych nawet w przypadku dużych zbiorów danych. Obsługa transakcji przez SQL Server gwarantuje spójność danych i możliwość przywrócenia stanu przed transakcją w przypadku wystąpienia błędów podczas obsługi danych przez warstwę serwerową.⁸ Zastosowanie relacyjnego modelu bazy danych pozwala na uporządkowane przechowywanie danych, ułatwiając zarządzanie nimi oraz zapewniając integralność danych poprzez unikatowe identyfikatory dla każdego rekordu oraz stosowanie kluczy głównych i obcych, co zapobiega powtórzeniom i gwarantuje spójność danych. Dzięki odpowiedniej optymalizacji zapytań, relacyjna baza danych jest w stanie szybko przetwarzać zapytania, nawet w przypadku dużych zbiorów danych. Niemniej jednak, wykorzystanie relacyjnego modelu bazy danych wymaga przestrzegania pewnych zaleceń, takich jak optymalizacja zapytań, regularne tworzenie kopii zapasowych oraz monitorowanie i utrzymanie wydajności systemu. Te czynniki należy uwzględnić, aby zapewnić efektywne funkcjonowanie systemu wspomagającego komunikację. Model bazy danych systemu został utworzony w myśl zasady rozbijania informacji na mniejsze użyteczne części, dzieląc je na tabele powiązane za pomocą relacji. Pełny model bazy danych dla systemu wspomagającego komunikację pomiędzy studentem a dziekanatem został przedstawiony na schemacie numer 9.

Centralnym elementem schematu encji jest tabela USER, odpowiedzialna za przechowywanie danych użytkowników systemu. Kluczem głównym tej tabeli jest kolumna id. Encja użytkowników zawiera informacje dotyczące logowania do systemu oraz dane administracyjne. Kolejne pola obejmują login (identyfikator użytkownika), password (hasło uwierzytelniające), last_login (data i czas ostatniego logowania), login_ban (informacja o możliwości logowania użytkownika), archive (status archiwalny użytkownika). Ponadto,

⁸ What is SQL Server?, <https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16>, Randolph West, Mike Ray, [14.02.2024]

tabela przechowuje także dane personalne, takie jak imię (forename) i nazwisko (surname). Encja użytkowników posiada także powiązania z obiektami grupującymi użytkowników na podstawie grupy użytkowników i jednostki organizacyjnej za pomocą kluczy obcych. Pole `group_id` jest używane do przypisania użytkownika do grupy w systemie i odnosi się do tabeli `USER_GROUP`, która jest słownikiem zawierającym informacje dotyczące dostępnych ról w systemie oraz określających uprawnienia użytkowników. Struktura słownika jest także stosowana w przypadku encji grup użytkowników, jednostek organizacyjnych, kategorii zgłoszenia oraz statusu zgłoszenia. Drugim kluczem obcym w tabeli użytkowników jest kolumna `organization_user_id`, która umożliwia przypisanie użytkownika do domyślnej jednostki operacyjnej organizacji i odnosi się do tabeli `ORGANIZATION_UNIT`.



Rysunek 9 Model relacyjnej bazy danych systemu wspierającego komunikację pomiędzy studentem a pracownikiem dziekanatu

Klucz główny encji użytkowników jest wykorzystywany w tabelach `TICKET` oraz `TICKET_MESSAGE`. Tabela `TICKET` jest odpowiedzialna za przechowywanie nagłówków zgłoszeń. Posiada unikalny klucz główny `id` oraz powiązania kluczy obcych do encji użytkowników poprzez kolumny `operator_id` (przypisujące osobę obsługującą zgłoszenie) i `student_id` (łączące drugą stronę komunikacji w wymianie informacji). Oprócz powiązań z encją użytkowników, tabela zgłoszeń zawiera powiązania z encjami słownikowymi. Powiązanie z encją kategorii zgłoszeń jest realizowane przez klucz obcy `category_id`, a z encją statusów zgłoszeń przez klucz obcy `status_id`. Dodatkowo, tabela zawiera kolumny `inserted_date` (data i czas utworzenia zgłoszenia), `change_date` (data i

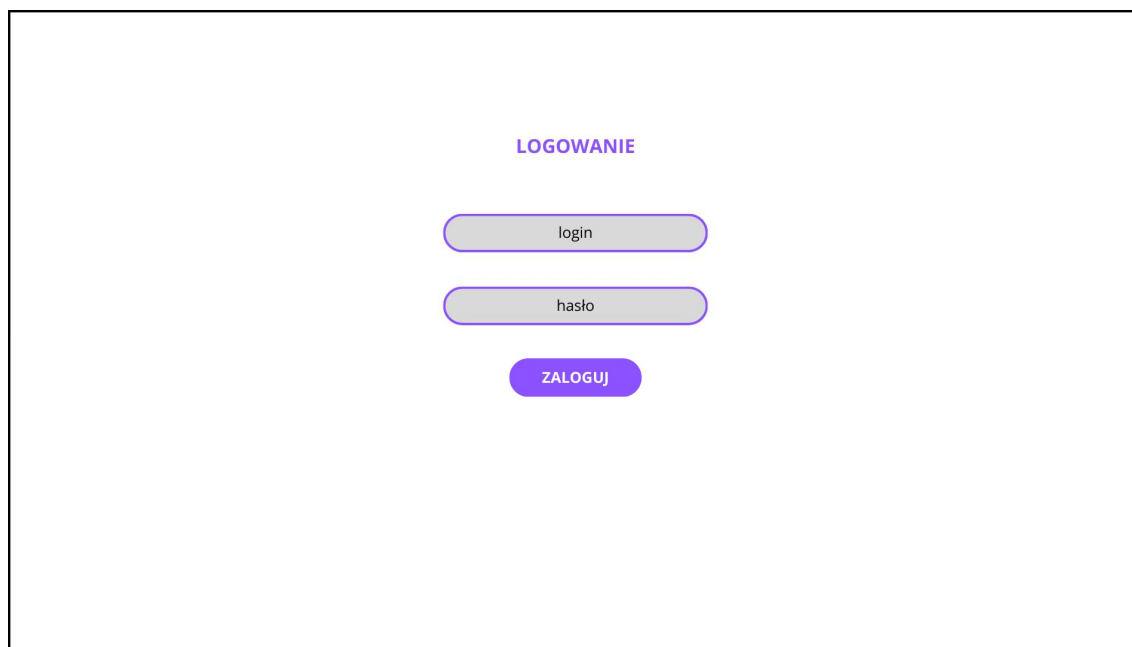
czas ostatniej zmiany w zgłoszeniu) oraz archive (określenie archiwalności zgłoszenia). Tabela TICKET_MESSAGE, oprócz powiązania z encją użytkownika, posiada również powiązanie z encją nagłówków zgłoszeń. Encja ta przechowuje wiadomości związane ze zgłoszeniami i posiada unikalny klucz główny id. Zawiera dwa klucze obce, ticket_id (odwołujący się do nagłówka zgłoszenia, w którym została napisana wiadomość) oraz sender_id (odwołujący się do użytkownika, który wysłał wiadomość w zgłoszeniu). Oprócz kolumn przechowujących klucze obce, tabela zawiera również kolumny message_text (tekst wiadomości), inserted_date (data i czas dodania wiadomości). Ostatnim elementem związanym z zgłoszeniami w modelu struktury bazy danych jest tabela MESSAGE_ATTACHMENT, która przechowuje załączniki przesyłane wraz z wiadomościami do zgłoszeń. Encja ta zawiera unikalny klucz główny id, klucz obcy message_id (łączy załącznik z wiadomością do zgłoszenia) oraz kolumnę attachment (przechowującą załącznik dołączony do wiadomości).

5.5 Projekt interfejsu użytkownika

W rozdziale tym przedstawione zostanie koncepcja elementów interfejsu użytkownika systemu wspomagającego wymianę wiadomości pomiędzy studentem a dziekanatem. Głównym założeniem definiującym projekt interfejsu jest jego prostota i łatwość użytkowania nakierowana na jego intuicyjną obsługę.

Zaprojektowane rozwiązanie obejmuje mechanizm logowania do systemu, który został przedstawiony na rysunku 10. Logowanie będzie możliwe za pomocą konta utworzonego przez administratora systemu. Aby administrator mógł tworzyć konta, będzie miał dostęp do specjalnego menu zarządzania użytkownikami, przedstawionego na rysunku 17. Menu administratora będzie dostępne z ekranu głównego administratora, zgodnie z rysunkiem 12, poprzez dodanie odpowiedniego przycisku otwierającego to menu, rozszerzając ekran główny udostępniony dla studentów i pracowników dziekanatu, przedstawiony na rysunku 11. Użytkownicy systemu będą mogli dodawać nowe zgłoszenia poprzez wypełnienie formularza pokazanego na rysunku 13. Dostęp do listy aktywnych i zaległych zgłoszeń będzie możliwy poprzez wybór odpowiedniego zgłoszenia z listy, która będzie dostępna w widoku, przedstawionym na rysunku 14. Formularz zgłoszenia będzie się różnił w zależności od jego statusu - aktywnego lub zamkniętego/zarchiwizowanego. Formularz aktywnego zgłoszenia, widoczny na rysunku 15, umożliwi dodanie nowej wiadomości z

załącznikiem oraz przegląd wcześniej wysłanych informacji i plików. Natomiast formularz zamkniętego lub zarchiwizowanego zgłoszenia, przedstawiony na rysunku 16, umożliwi jedynie przegląd informacji i plików przekazanych w ramach zgłoszenia.



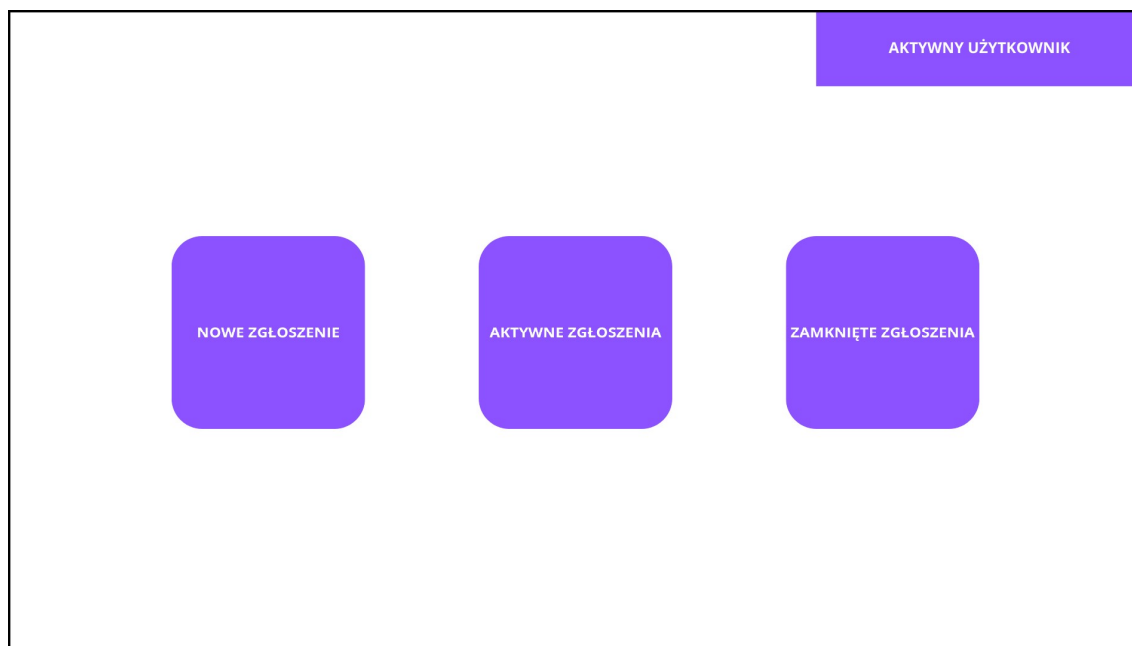
LOGOWANIE

login

hasło

ZALOGUJ

Rysunek 10 Projekt ekranu logowania do systemu.



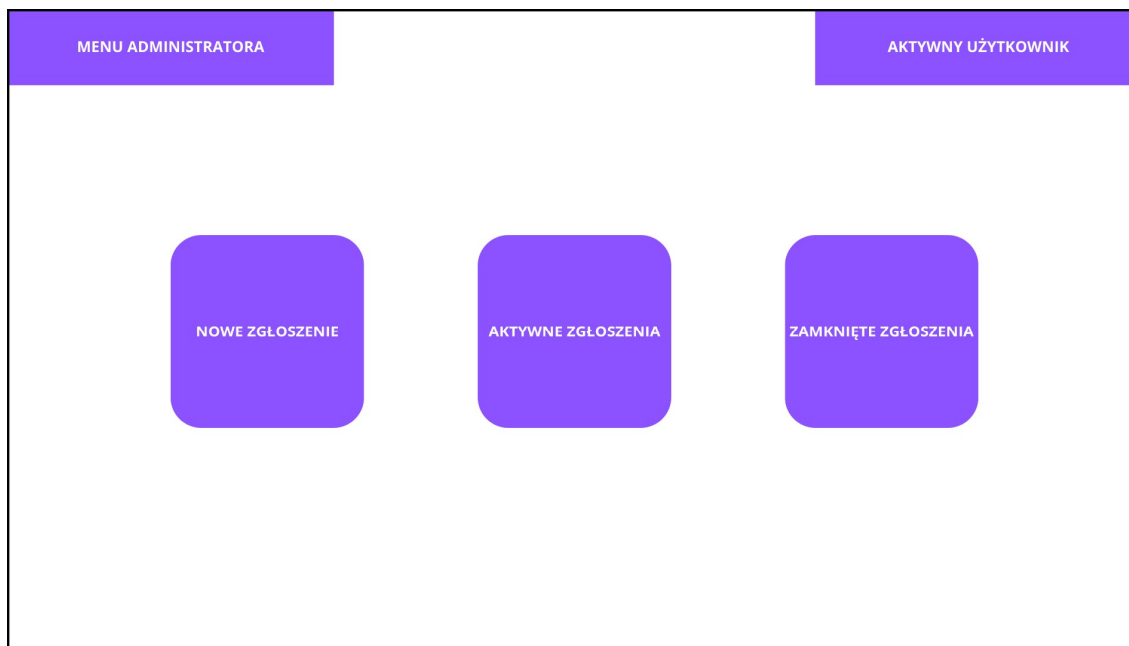
AKTYWNY UŻYTKOWNIK

NOWE ZGŁOSZENIE

AKTYWNE ZGŁOSZENIA

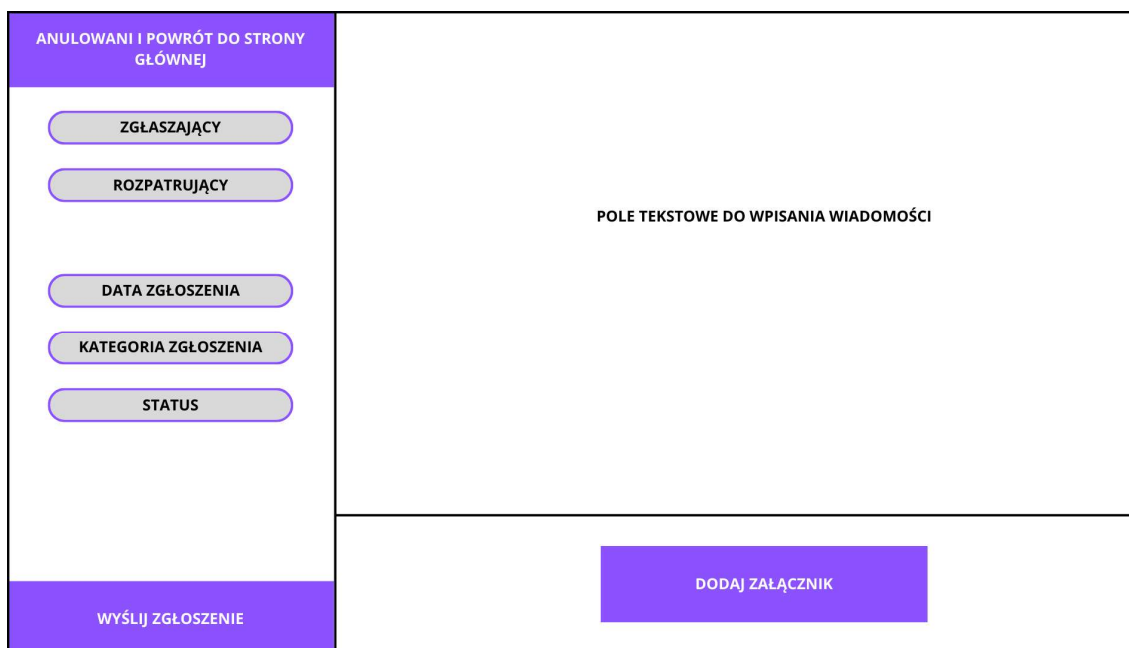
ZAMKNIĘTE ZGŁOSZENIA

Rysunek 11 Projekt ekranu głównego systemu dla studenta i pracownika dziekanatu.



The image shows a main menu for an administrator. It has a white background with a light blue header bar. The header bar contains two buttons: 'MENU ADMINISTRATORA' on the left and 'AKTYWNY UŻYTKOWNIK' on the right. Below the header bar, there are three large, rounded rectangular buttons arranged horizontally. From left to right, they are labeled 'NOWE ZGŁOSZENIE', 'AKTYWNE ZGŁOSZENIA', and 'ZAMKNIĘTE ZGŁOSZENIA'.

Rysunek 12 Projekt ekranu głównego systemu dla administratora systemu



The image shows a form for creating a new report. It has a white background with a light blue header bar. The header bar contains the text 'ANULOWANI I POWRÓT DO STRONY GŁÓWNEJ'. Below the header bar, there is a vertical list of five buttons: 'ZGŁASZAJĄCY', 'ROZPATRUJĄCY', 'DATA ZGŁOSZENIA', 'KATEGORIA ZGŁOSZENIA', and 'STATUS'. To the right of these buttons is a large text area labeled 'POLE TEKSTOWE DO WPISANIA WIADOMOŚCI'. At the bottom of the form, there is a button labeled 'WYŚLIJ ZGŁOSZENIE' on the left and a button labeled 'DODAJ ZAŁĄCZNIK' on the right.

Rysunek 13 Formularz utworzenia nowego zgłoszenia

POWRÓT NA STRONĘ GŁÓWNĄ					FILTR	
NAGŁÓWEK 1	NAGŁÓWEK 2	NAGŁÓWEK 3	NAGŁÓWEK 4	NAGŁÓWEK 5	NAGŁÓWEK 6	NAGŁÓWEK 7
REKORD 1						
REKORD 2						
REKORD 3						
REKORD 4						
REKORD 5						

Rysunek 14 Formatka dla podglądu aktywnych i zamkniętych zgłoszeń

POWRÓT DO ZGŁOSZEŃ

ZGŁASZAJĄCY

ROZPATRUJĄCY

DATA ZGŁOSZENIA

KATEGORIA ZGŁOSZENIA

STATUS

WYŚLIJ WIADOMOŚĆ

POLE NA WPISANIE WIADOMOŚCI

DODAJ ZAŁĄCZNIK

LISTA ZAŁĄCZNIKÓW DO WIADOMOŚCI

WIADOMOŚCI AKTYWNEGO UŻYTKOWNIKA

WIADOMOŚCI DRUGIEGO UŻYTKOWNIKA

ZAŁĄCZNIK

ZAŁĄCZNIK

ZAŁĄCZNIK

ZAŁĄCZNIK

ZAŁĄCZNIK

Rysunek 15 Formularz podglądu i wysłania wiadomości w ramach istniejącego zgłoszenia

Rysunek 16 Formularz podglądu zamkniętego lub zarchiwizowanego zgłoszenia

POWRÓT NA STRONĘ GŁÓWNA		FILTR		OPERACJA
LOGIN	IMIĘ	NAZWISKO	DATA OSTATNIEGO LOGOWANIA	ARCHIWALNY
UŻYTKOWNIK 1	TESTOWY	JAN	24.03.2024 07:45:13	NIE
UŻYTKOWNIK 2				
UŻYTKOWNIK 3				
UŻYTKOWNIK 4				
UŻYTKOWNIK 5				

Rysunek 17 Widok menu dostępnego dla administratora systemu

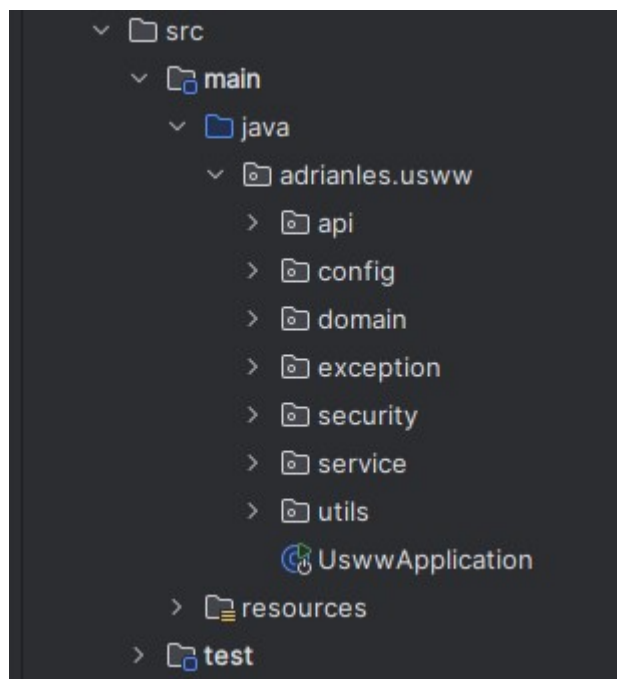
6 Dokumentacja techniczna

System został zaprojektowany w sposób umożliwiający podział na dwie kluczowe warstwy: logikę biznesową oraz warstwę prezentacji. Taki podział zapewnia łatwiejszy rozwój i obsługę aplikacji poprzez separację odpowiedzialności i umożliwia skuteczne rozwiązywanie problemów w poszczególnych komponentach systemu. Ten podejście ma istotny wpływ na skalowalność oraz możliwość wsparcia aplikacji. Zgodnie z ustaleniami, warstwa logiki biznesowej została zaimplementowana z wykorzystaniem architektury REST API, co umożliwia komunikację między różnymi komponentami systemu oraz zapewnia elastyczność w budowie aplikacji. Natomiast warstwa prezentacji została zrealizowana za pomocą klienta opartego na szablonie Angular, co pozwala na dynamiczne wyświetlanie danych oraz interakcję użytkownika z systemem poprzez interfejs graficzny.

6.1 Struktura warstwy logiki biznesowej

Struktura projektu warstwy logiki biznesowej, przedstawiona na rysunku 18 została starannie zaprojektowana z myślą o zapewnieniu modularności oraz czytelności kodu źródłowego architektury aplikacji. Aby osiągnąć ten cel, architektura została podzielona na pakiety, które zostały nazwane w sposób opisujący zadania i zakres odpowiedzialności, za które odpowiadają klasy w danym pakiecie. Dzięki temu podejściu, można łatwiej poruszać się po kodzie, zrozumieć jego strukturę oraz szybciej wdrażać poprawki i dodawać nowe funkcjonalności. Dodatkowo, modułowy charakter architektury pozwala na łatwe rozszerzanie funkcjonalności poprzez dodanie nowych modułów lub modyfikację istniejących, co przyczynia się do elastyczności oraz skalowalności systemu⁹.

⁹ Rozdział V. Architektura, „Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów”, R. C. Martin, 11.05.2018, str. 151



Rysunek 18 Struktura projektu warstwy logiki biznesowej

6.1.1 Główny pakiet systemu

Główny pakiet aplikacji *usww* zawiera klasę startową oznaczoną adnotacją *@SpringBootApplication*, która stanowi kluczowy element inicjalizacji frameworka Spring Boot¹⁰. Ta adnotacja jest skróconym zapisem trzech fundamentalnych adnotacji: *@EnableAutoConfiguration*, *@ComponentScan* oraz *@Configuration*.

Adnotacja *@EnableAutoConfiguration* uruchamia mechanizm automatycznej konfiguracji, który na podstawie zależności dodanych do projektu oraz dostępnych klas w parametrze położenia klas i pakietów konfiguruje różne komponenty aplikacji. Eliminuje to potrzebę ręcznego definiowania wielu elementów konfiguracyjnych, przyspieszając rozwój aplikacji.

Adnotacja *@ComponentScan* odpowiada za wyszukiwanie klas oznaczonych adnotacjami stereotypowymi takimi jak *@Component*, *@Service*, *@Repository*, *@Controller* czy *@RestController*. Domyślnie skanowanie obejmuje pakiet, w którym znajduje się klasa startowa oraz wszystkie jego podpakiety. Szablon Spring automatycznie tworzy instancje znalezionych komponentów i zarządza ich cyklem życia.

¹⁰ Using the *@SpringBootApplication* Annotation, <https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html>

Adnotacja *@Configuration* wskazuje, że klasa zawiera definicje konfiguracyjne, najczęściej w postaci metod oznaczonych adnotacją *@Bean*. Pozwala to na programowe tworzenie i konfigurowanie komponentów oraz na importowanie innych klas konfiguracyjnych.

Dzięki zastosowaniu *@SpringBootApplication* w klasie startowej, aplikacja zyskuje zaawansowane mechanizmy auto-konfiguracji i zarządzania komponentami, co istotnie upraszcza rozwój i utrzymanie oprogramowania.

6.1.2 Pakiet konfiguracyjny

W pakiecie *config* znajdują się wszystkie klasy oznaczone adnotacją *@Configuration*, które są odpowiedzialne za konfigurację różnych części aplikacji. Przykładem wykorzystania tej adnotacji jest klasa *DataSourceConfig*, której zadaniem jest konfiguracja źródła danych aplikacji. W załączonym kodzie źródłowym klasy, znajduje się również adnotacja *@Bean*, która opatruje metodę *getDataSource()*. Metody oznaczone tą adnotacją są odpowiedzialne za tworzenie i konfigurowanie obiektów, które są zarządzane przez szablon Spring.

W pakiecie *config* znajdują się klasy oznaczone adnotacją *@Configuration*, które definiują komponenty infrastrukturalne aplikacji i stanowią kluczowy element konfiguracji kontekstu Spring. Adnotacja *@Configuration* oznacza, że dana klasa zawiera definicje *@Bean*, czyli obiektów zarządzanych przez kontekst aplikacji Spring.

Przykładem takiej klasy jest *DataSourceConfig*, odpowiedzialna za konfigurację źródła danych (*DataSource*), czyli obiektu zapewniającego połączenie z bazą danych. W klasie tej znajduje się metoda *getDataSource()* oznaczona adnotacją *@Bean*. Metody opatrzone tą adnotacją informują Spring Framework, że zwracane przez nie obiekty powinny zostać zarejestrowane w kontekście aplikacji jako komponenty dostępne do wstrzykiwania w innych częściach systemu.

Podczas uruchamiania aplikacji, Spring automatycznie odnajduje wszystkie klasy oznaczone jako *@Configuration*, przetwarza metody oznaczone jako *@Bean* i rejestruje zwrócone obiekty w kontekście aplikacji. Dzięki temu mechanizmowi możliwe jest centralne zarządzanie konfiguracją oraz wykorzystanie systemu wstrzykiwania zależności do przekazywania skonfigurowanych obiektów do innych komponentów systemu, które ich potrzebują.

```

@Configuration
@EnableTransactionManagement
public class DataSourceConfig {

    @Bean
    public DataSource getDataSource() {
        HikariConfig hikariConfig = new HikariConfig();
        setDatabaseProperties(hikariConfig);
        setHikariProperties(hikariConfig);
        HikariDataSource dataSource = new HikariDataSource(hikariConfig);
        executeSqlScripts(dataSource);
        return dataSource;
    }
}

```

6.1.3 Pakiet modelu danych i repozytoriów

Pakiet *domain* stanowi warstwę dostępu do danych aplikacji i zawiera elementy odpowiedzialne za mapowanie obiektowo-relacyjne oraz interakcję z bazą danych. W ramach tego pakietu wyodrębniono dwa główne komponenty: pakiet *entity* z klasami reprezentującymi encje oraz pakiet *repository* zawierający interfejsy dostępu do danych.

W pakiecie *entity* znajdują się klasy oznaczone adnotacją *@Entity* z pakietu *jakarta.persistence*. Adnotacja ta wskazuje, że klasa reprezentuje encję, którą szablon JPA mapuje na odpowiednią tabelę w bazie danych. Przykładem jest klasa *User*, rozszerzająca abstrakcyjną klasę *AbstractEntity* zawierającą wspólne atrybuty dla wszystkich encji, jak identyfikator.

```

package adrianles.usww.domain.entity;

import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.MappedSuperclass;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.io.Serializable;

@MappedSuperclass
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public abstract class AbstractEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
}

```

Aby wyeliminować powtarzalny kod (boilerplate code) związany z metodami dostępowymi, konstruktorami i innymi standardowymi elementami klas, w projekcie zastosowano bibliotekę Lombok. Adnotacje `@Getter` i `@Setter` automatycznie generują metody dostępowe do pól (akcesory i mutatory). Natomiast adnotacje `@NoArgsConstructor` i `@AllArgsConstructor` odpowiadają za utworzenie konstruktorów: bezparametrowego oraz przyjmującego wszystkie pola klasy jako parametry.

W pakiecie *repository* zdefiniowano interfejsy rozszerzające *JpaRepository*, które zapewniają standardowe operacje na danych (CRUD) oraz umożliwiają definiowanie niestandardowych zapytań. Spring Data JPA, bazując na tych interfejsach, automatycznie implementuje metody dostępu do danych, eliminując konieczność ręcznego pisania kodu SQL. Przykładem interfejsu rozszerzającego *JpaRepository* jest interfejs *UserRepository*.

```
package adrianles.usww.domain.repository;

import adrianles.usww.domain.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
    Optional<User> findByLogin(String login);

    List<User> findAllByArchiveTrue();
}
```

6.1.4 Pakiet logiki biznesowej

W pakiecie *service* znajdują się klasy odpowiedzialne za logikę biznesową aplikacji, które zgodnie z konwencją Spring Framework, oznaczone są adnotacją `@Service`. Zastosowano podejście oparte na interfejsach i ich implementacjach, wykorzystując wzorzec projektowy fasady¹¹, gdzie interfejsy usług zdefiniowane są w podpakiecie *service.facade*, a ich konkretne implementacje w *service.impl*. Serwisy stanowią warstwę pośrednią między kontrolerami a repozytoriami, enkapsulując logikę biznesową i operacje na danych. Wiele interfejsów serwisowych oznaczonych jest dodatkowo adnotacją `@Transactional`, co

¹¹ Fasada (Facade), „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku”, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Helion, 2021, str. 161

zapewnia integralność transakcji bazodanowych. Przykładem takiej klasy jest *TicketServiceImpl*, które implementują odpowiednie interfejsy i realizują określone funkcjonalności biznesowe, zachowując zasadę pojedynczej odpowiedzialności według metodyki SOLID.

```
@Service
@RequiredArgsConstructor
public class TicketServiceImpl implements TicketService {
    private final TicketRepository ticketRepository;
    private final UserRepository userRepository;
    private final TicketCategoryRepository ticketCategoryRepository;
    private final TicketStatusRepository ticketStatusRepository;
    private final TicketPriorityRepository ticketPriorityRepository;
    private final TicketMapper ticketMapper;

    @Override
    public List<TicketDTO> getAllTickets() {
        return ticketRepository.findAll().stream()
            .map(ticketMapper::toDto)
            .collect(Collectors.toList());
    }

    @Override
    public Page<TicketDTO> getFilteredTickets(TicketFilterCriteriaDTO
criteria, Pageable pageable) {
        Specification<Ticket> spec =
TicketSpecifications.buildSpecification(criteria);
        return ticketRepository.findAll(spec,
pageable).map(ticketMapper::toDto);
    }

    @Override
    public TicketDTO getTicketById(Integer id) {
        Ticket ticket = findTicketById(id);
        return ticketMapper.toDto(ticket);
    }
}
```

6.1.5 Pakiet kontrolerów REST API

W pakiecie *api* umieszczone są komponenty odpowiedzialne za komunikację zewnętrzną i prezentację danych aplikacji, tworząc kompletną warstwę prezentacji API REST. Struktura pakietu podzielona jest na logiczne podpakiety: *controller*, *dto* i *mapper*. Kontrolery, oznaczone adnotacją *@RestController*, obsługują żądania HTTP i definiują punkty końcowe API, co widać w klasach jak *TicketController*, *UserController* czy *AuthController*.

```

package adrianles.usww.api.controller.common;

import adrianles.usww.api.dto.UserDTO;
import adrianles.usww.service.facade.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @GetMapping("/profile")
    public ResponseEntity<UserDTO> getCurrentUserProfile() {
        Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();
        String currentUsername = authentication.getName();
        return ResponseEntity.ok(
            userService.getUserByLogin(currentUsername));
    }

    @GetMapping("/{id}/basic-info")
    public ResponseEntity<UserDTO> getUserBasicInfo(@PathVariable int id) {
        return ResponseEntity.ok(userService.getUserBasicInfo(id));
    }
}

```

Dodatkowo, pakiet *dto* zawiera obiekty transferu danych (DTO), które służą jako kontenery do bezpiecznej wymiany informacji między warstwą prezentacji a logiką biznesową, eliminując bezpośrednią ekspozycję encji domeny.

Pakiet *mapper* z kolei zawiera klasy odpowiedzialne za transformację obiektów między reprezentacjami DTO a encjami domenowymi, co widać w implementacjach takich jak *UserMapper* czy *TicketMapper*.

Taka organizacja wspiera zasadę rozdzielania odpowiedzialności i zwiększa możliwości utrzymania i rozszerzania systemu.

```
package adrianles.usww.api.mapper;

import adrianles.usww.api.dto.UserDTO;
import adrianles.usww.domain.entity.User;
import org.springframework.stereotype.Component;

@Component
public class UserMapper {

    public UserDTO toDto(User user) {
        if (user == null) {
            return null;
        }

        UserDTO userDTO = new UserDTO();
        userDTO.setId(user.getId());
        userDTO.setLogin(user.getLogin());
        userDTO.setForename(user.getForename());
        userDTO.setSurname(user.getSurname());
        userDTO.setArchive(user.isArchive());
        userDTO.setFirstLogin(user.isFirstLogin());
        userDTO.setLoginBan(user.isLoginBan());

        String lastLogin = user.getLastLogin() != null ?
            user.getLastLogin().toString() : null;
        userDTO.setLastLogin(lastLogin);

        Integer userGroupId = user.getUserGroup() != null ?
            user.getUserGroup().getId() : null;
        userDTO.setGroupId(userGroupId);

        Integer organizationUnitId = user.getOrganizationUnit() != null ?
            user.getOrganizationUnit().getId() : null;
        userDTO.setOrganizationUnitId(organizationUnitId);

        return userDTO;
    }

    public UserDTO toBasicInfoDto(User user) {
        if (user == null) {
            return null;
        }

        UserDTO basicInfo = new UserDTO();
        basicInfo.setId(user.getId());
        basicInfo.setForename(user.getForename());
        basicInfo.setSurname(user.getSurname());

        return basicInfo;
    }
}
```

6.1.6 Pakiet komponentów bezpieczeństwa

W pakiecie *security* znajdują się komponenty odpowiedzialne za zabezpieczenia aplikacji, które realizują mechanizmy uwierzytelniania i autoryzacji użytkowników. Struktura tego pakietu obejmuje dwa główne podpakiety: *jwt* oraz *userdetails*.

Podpakiet *jwt* zawiera klasy odpowiedzialne za implementację mechanizmu JSON Web Token, w tym *JwtUtil* do generowania, walidacji i ekstrakcji danych z tokenów oraz *JwtAuthenticationFilter*, który przechwytuje i przetwarza żądania HTTP w celu weryfikacji tokenów JWT.

Podpakiet *userdetails* implementuje i rozszerza standardowy mechanizm Spring Security, dostarczając klasę *UserDetailsServiceImpl*, która odpowiada za ładowanie danych użytkowników z bazy danych, oraz klasę *CustomUserDetails* wraz z interfejsem *ExtendedUserDetails*, które wzbogacają standardowe informacje o użytkowniku o dodatkowe atrybuty i funkcjonalności, takie jak sprawdzanie pierwszego logowania czy rozróżnianie ról użytkowników w systemie.

6.1.7 Pakiet obsługi wyjątków

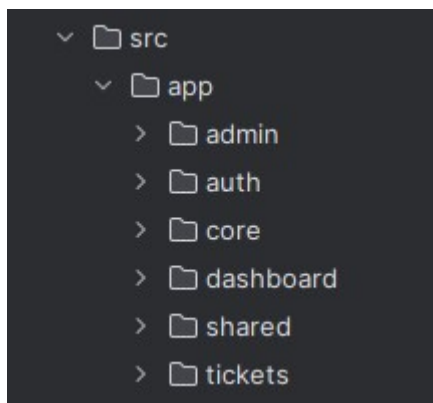
W pakiecie *exception* ulokowane są klasy i interfejsy odpowiedzialne za obsługę sytuacji wyjątkowych w aplikacji. Pakiet ten zawiera zarówno definicje niestandardowych wyjątków, jak klasę *ResourceNotFoundException*, oznaczoną adnotacją *@ResponseStatus* dla automatycznego mapowania na odpowiedni kod HTTP, jak i mechanizmy ich przechwytywania.

W podpakiecie *handler* zastosowano wzorzec projektowy łańcucha zobowiązań¹², widoczny poprzez interfejs *ExceptionHandler* oraz abstrakcyjną implementację *AbstractExceptionHandler*, które tworzą łańcuch obsługi różnych typów wyjątków. Klasa *GlobalExceptionHandler*, oznaczona adnotacją *@ControllerAdvice*, pełni rolę centralnego punktu przechwytywania wyjątków w całej aplikacji, delegując ich obsługę do odpowiednich wyspecjalizowanych obiektów przechowujących. Taka architektura zapewnia jednolity i spójny mechanizm obsługi błędów, przekształcając wyjątki aplikacyjne na ustrukturyzowane i zrozumiałe dla klienta odpowiedzi HTTP, co poprawia zarówno jakość, jak i czytelność komunikacji z interfejsem użytkownika.

¹² Łańcuch zobowiązań (Chain of responsibility), „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku”, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Helion, 2021, str. 244

6.2 Struktura warstwy prezentacji

Struktura projektu warstwy prezentacji, przedstawiona na rysunku 19 została zaprojektowana z myślą o zapewnieniu optymalnej organizacji interfejsu użytkownika oraz efektywnej komunikacji z warstwą logiki biznesowej. Architektura została starannie podzielona na modułowe komponenty zgodnie z konwencjami szablonu Angular, co zapewnia przejrzystość i łatwość utrzymania kodu. Poszczególne moduły funkcjonalne, takie jak *auth*, *dashboard*, *tickets* czy *admin*, grupują komponenty o pokrewnej funkcjonalności, co ułatwia nawigację w kodzie źródłowym oraz sprawne wprowadzanie modyfikacji. Zastosowanie architektury komponentowej Angular pozwala na enkapsulację logiki i widoku poszczególnych części aplikacji, co zwiększa możliwość wielokrotnego wykorzystania kodu oraz umożliwia niezależny rozwój poszczególnych funkcjonalności. Reaktywne podejście do zarządzania stanem aplikacji poprzez usługi Angulara zapewnia efektywną komunikację z API oraz spójny przepływ danych w aplikacji. Dodatkowo, wykorzystanie języka TypeScript zwiększa bezpieczeństwo typów oraz ułatwia refaktoryzację kodu, co przyczynia się do ogólnej jakości i stabilności aplikacji.



Rysunek 19 Struktura projektu warstwy prezentacji

Moduł *auth* zawiera komponenty odpowiedzialne za procesy uwierzytelniania i autoryzacji użytkowników. Obejmuje komponenty logowania, zmiany hasła, usługę zarządzania tokenami JWT oraz interceptor, który automatycznie dodaje token uwierzytelniający do żądań HTTP.

Moduł *dashboard* implementuje główny panel informacyjny aplikacji, prezentujący zbiorcze statystyki i aktualności. Wyświetla najnowsze zgłoszenia, ich statusy oraz umożliwia szybki dostęp do najczęściej używanych funkcji systemu.

Moduł *tickets* składa się z komponentów do kompleksowej obsługi zgłoszeń. Zawiera widoki listy zgłoszeń z funkcjami filtrowania i sortowania, formularz tworzenia nowych zgłoszeń, widok szczegółów zgłoszenia z historią wiadomości oraz system dodawania załączników.

Moduł *admin* grupuje komponenty zarządzania systemem dostępne dla administratorów. Zawiera funkcje zarządzania użytkownikami (dodawanie, edycja, blokowanie), konfiguracji parametrów systemu oraz przeglądania działań użytkowników.

Moduł *shared* przechowuje uniwersalne komponenty wielokrotnego użytku, takie jak układ elementów, nagłówki, stopka i menu boczne. Obejmuje również wspólne usługi, w tym usługę komunikacji z API słownikowym oraz komponenty UI używane w wielu miejscach aplikacji.

Komponenty główne na najwyższym poziomie znajdują się komponenty zarządzające routinguem, konfiguracją aplikacji oraz definiujące główną strukturę interfejsu użytkownika.

7 Testy i weryfikacja systemu

W procesie tworzenia systemu wspomagającego komunikację między studentami a dziekanatem, kompleksowe testowanie odgrywało kluczową rolę w zapewnieniu jakości, niezawodności i poprawności działania aplikacji. Testowanie zostało zintegrowane z cyklem rozwoju oprogramowania, co pozwoliło na wczesne wykrywanie i eliminowanie defektów. W trakcie procesu implementacji zastosowano różnorodne techniki i poziomy testowania, zapewniając tym samym wielowarstwową weryfikację systemu przed jego wdrożeniem.

Proces weryfikacji systemu obejmował zarówno automatyczne, jak i manualne metody testowania, które wspólnie zapewniły wysoką jakość dostarczonego rozwiązania. Szczególną uwagę poświęcono obszarom krytycznym aplikacji, takim jak uwierzytelnianie, zarządzanie zgłoszeniami czy komunikacja między warstwami systemu. Podczas testowania udało się zidentyfikować szereg problemów, które zostały następnie rozwiązane, co znacząco poprawiło końcową jakość systemu.

7.1 Testy jednostkowe

W ramach implementacji kompleksowego procesu testowania systemu USWW, zastosowano systematyczne podejście do weryfikacji poszczególnych komponentów aplikacji poprzez testy jednostkowe. Proces testowania jednostkowego został zrealizowany przy wykorzystaniu następującego zestawu narzędzi:

- JUnit 5 (Jupiter) - stanowiącego fundamentalne środowisko wykonawcze testów,
- Mockito - szablon umożliwiającego efektywną izolację jednostek testowych poprzez symulowanie zależności zewnętrznych,
- AssertJ - biblioteki asercji zapewniającej czytelną i ekspresyjną składnię weryfikacji rezultatów.

7.1.1 Struktura organizacyjna testów

W ramach projektu implementacyjnego zastosowano rygorystyczne podejście do organizacji testów jednostkowych, uwzględniające modułową strukturę aplikacji. Testy zostały ustrukturyzowane w pakietach, których nazewnictwo odzwierciedla architekturę badanego systemu, co zapewnia przejrzystość oraz

efektywne zarządzanie procesem testowania. Struktura pakietów testowych obejmuje następujące kategorie:

- *unit.api.controller* - zawierający testy kontrolerów obsługujących żądania HTTP, weryfikujących poprawność obsługi punktów końcowych REST API,
- *unit.api.mapper* - grupujący testy klas odpowiedzialnych za transformację obiektów między warstwami systemu,
- *unit.domain.entity* - obejmujący testy weryfikujące poprawność implementacji modeli danych oraz ich adnotacji,
- *unit.domain.specification* - zawierający testy mechanizmów dynamicznego budowania zapytań bazodanowych,
- *unit.exception* - dedykowany do testowania komponentów obsługi sytuacji wyjątkowych,
- *unit.security* - weryfikujący funkcjonalność mechanizmów bezpieczeństwa, w tym uwierzytelniania i autoryzacji,
- *unit.service* - obejmujący testy warstwy usług zawierającej logikę biznesową systemu,
- *unit.utils* - grupujący testy klas narzędziowych realizujących funkcjonalności pomocnicze.

Przyjęta struktura organizacyjna testów zapewnia systematyczne podejście do procesu weryfikacji, umożliwiając skuteczną identyfikację oraz izolację potencjalnych defektów w poszczególnych komponentach aplikacji. Każdy pakiet testowy koncentruje się na weryfikacji określonego aspektu funkcjonalnego systemu, co przyczynia się do kompleksowej oceny jakości implementacji.

7.1.2 Metodyka testowania warstwy serwisowej

Warstwa serwisowa systemu USWW, zawierająca kluczową logikę biznesową aplikacji, została objęta kompleksowym procesem testowym uwzględniającym następujące zasady metodologiczne.

Każda jednostka testowa została izolowana poprzez zastosowanie techniki mockowania zależności zewnętrznych. Podejście to umożliwiło koncentrację na testowanej funkcjonalności bez wpływu czynników zewnętrznych, co przyczyniło się do zwiększenia wiarygodności wyników testów.

Przeprowadzone testy obejmują zarówno pozytywne, jak i negatywne ścieżki wykonania kodu, co zapewnia kompleksową weryfikację zachowania systemu w

różnorodnych sytuacjach operacyjnych. Szczególną uwagę poświęcono testowaniu reakcji systemu na niepoprawne dane wejściowe.

Testy warstwy serwisowej uwzględniają szczegółową weryfikację poprawności transformacji danych między poszczególnymi warstwami aplikacji, co stanowi kluczowy aspekt integralności danych w systemie.

Metodyka testowania obejmuje systematyczną weryfikację zachowania systemu w przypadkach granicznych oraz podczas obsługi wyjątków, co zwiększa odporność aplikacji na nieprzewidziane sytuacje w środowisku produkcyjnym.

Implementacja testów warstwy serwisowej opiera się na wzorcu AAA (Arrange-Act-Assert), co ilustruje następujący przykład:

```
@Test
@DisplayName("Powinien utworzyć nowego użytkownika")
void createUser_shouldCreateNewUser() {
    // given
    UserDTO newUserDTO = new UserDTO();
    newUserDTO.setLogin("newuser");
    newUserDTO.setForename("Anna");
    newUserDTO.setSurname("Nowak");
    newUserDTO.setGroupId(1);
    newUserDTO.setOrganizationUnitId(1);

    // Konfiguracja zachowania mocków
    when(userGroupRepository.findById(1))
        .thenReturn(Optional.of(adminGroup));
    when(organizationUnitRepository.findById(1))
        .thenReturn(Optional.of(organizationUnit));
    when(passwordEncoder.encode("newuser"))
        .thenReturn("encodedPassword");
    when(userRepository.save(any(User.class)))
        .thenReturn(savedUser);
    when(userMapper.toDto(savedUser))
        .thenReturn(savedUserDTO);

    // when
    UserDTO result = userService.createUser(newUserDTO);

    // then
    assertThat(result).isEqualTo(savedUserDTO);
    assertThat(result.getGeneratedPassword())
        .isEqualTo("newuser");

    // Weryfikacja interakcji
    verify(userGroupRepository, times(2)).findById(1);
    verify(organizationUnitRepository).findById(1);
    verify(passwordEncoder).encode("newuser");
    verify(userRepository).save(any(User.class));
    verify(userMapper).toDto(savedUser);
}
```

Istotnym elementem metodyki testowej jest również weryfikacja poprawności obsługi przypadków negatywnych, co zwiększa niezawodność systemu w warunkach rzeczywistej eksploatacji. Poniższy przykład ilustruje test przypadku negatywnego, gdy system odpowiednio reaguje na próbę utworzenia użytkownika z nieistniejącą grupą:

```
@Test
@DisplayName("Powinien rzucić wyjątek gdy grupa użytkownika nie istnieje")
void createUser_shouldThrowExceptionWhenUserGroupNotFound() {
    // given
    UserDTO newUserDTO = new UserDTO();
    newUserDTO.setLogin("newuser");
    newUserDTO.setGroupId(99); // Nieistniejąca grupa

    when(userGroupRepository.findById(99)).thenReturn(Optional.empty());

    // when / then
    assertThatThrownBy(() -> userService.createUser(newUserDTO))
        .assertInstanceOf(ResourceNotFoundException.class)
        .hasMessageContaining("User group 99 does not exist");

    verify(userGroupRepository).findById(99);
    verifyNoInteractions(passwordEncoder, userRepository, userMapper);
}
```

7.1.3 Testowanie komponentów dostępu do danych

Komponenty dostępu do danych, a w szczególności specyfikacje JPA, stanowią krytyczny element aplikacji odpowiedzialny za efektywne pobieranie i manipulowanie danymi. Proces testowania tych komponentów skupił się na weryfikacji wielu kluczowych aspektów funkcjonalnych. Szczególną uwagę poświęcono poprawności konstrukcji dynamicznych zapytań, weryfikując czy kryteria wyszukiwania są prawidłowo budowane na podstawie przekazanych parametrów. Istotnym elementem procesu testowego była również analiza obsługi warunków brzegowych, gdzie zweryfikowano zachowanie systemu przy ekstremalnych wartościach parametrów, danych pustych oraz nieprawidłowych formatach. Testowanie objęło także precyzję filtrowania danych, sprawdzając dokładność i efektywność mechanizmów filtrowania oraz sortowania danych na podstawie wielokryterialnych specyfikacji. Dodatkowo przeprowadzono weryfikację integralności mapowania relacyjno-obiektowego, koncentrując się na poprawności przekształceń między strukturami bazy danych a modelami obiektowymi. Testy specyfikacji wyszukiwania dostarczyły nie tylko potwierdzenia poprawności działania komponentów, ale również umożliwiły

identyfikację potencjalnych wąskich gardeł wydajnościowych w procesie przetwarzania zapytań złożonych.

```
@Test
@DisplayName("buildSpecification powinien obsługiwać kryteria dotyczące pól")
void buildSpecification_shouldHandleCriteriaWithValidFields() {
    TicketFilterCriteriaDTO criteria = new TicketFilterCriteriaDTO();
    criteria.setTitle("test");
    criteria.setStudentId(1);
    criteria.setOperatorId(2);
    criteria.setStatusId(3);
    criteria.setCategoryId(4);
    criteria.setPriorityId(5);
    criteria.setArchive(true);

    Specification<Ticket> result =
        TicketSpecifications.buildSpecification(criteria);
    assertThat(result).isNotNull();
}
```

7.1.4 Testowanie warstwy mapperów

Warstwa mapperów pełni funkcję krytycznego interfejsu między modelami bazodanowymi a obiektami transferu danych (DTO), stanowiąc element architektury komunikacyjnej systemu. Proces testowania tej warstwy skoncentrował się na weryfikacji poprawności transformacji danych między różnymi reprezentacjami obiektowymi. Testy obejmowały analizę integralności kopiowania wartości atrybutów między obiektami domeny a odpowiadającymi im strukturami DTO. Procedury testowe uwzględniały weryfikację obsługi wartości nieokreślonej oraz referencji do obiektów powiązanych. Istotnym komponentem procesu walidacji była także weryfikacja poprawności transformacji typów danych specjalistycznych, takich jak daty, wartości wyliczeniowe czy struktury zagnieżdżone. Implementacja kompleksowych testów warstwy mapperów przyczyniła się do zapewnienia spójności danych w całym systemie oraz zminimalizowania ryzyka wystąpienia błędów związanych z nieprawidłową transformacją danych między warstwami aplikacji.

```
@Test
@DisplayName("toDto powinien poprawnie mapować pełny obiekt User na
UserDTO")
void toDto shouldMapFullUserToUserDTO() {
    // When
    UserDTO userDTO = userMapper.toDto(user);

    // Then
    assertThat(userDTO).isNotNull();
    assertThat(userDTO.getId()).isEqualTo(1);
    assertThat(userDTO.getLogin()).isEqualTo("testuser");
    assertThat(userDTO.getForename()).isEqualTo("Jan");
    assertThat(userDTO.getSurname()).isEqualTo("Kowalski");
    assertThat(userDTO.isLoginBan()).isFalse();
    assertThat(userDTO.getLastLogin()).isEqualTo("2024-02-16T10:00");
    assertThat(userDTO.getGroupId()).isEqualTo(1);
    assertThat(userDTO.getOrganizationUnitId()).isEqualTo(2);
    assertThat(userDTO.isArchive()).isFalse();
    assertThat(userDTO.isFirstLogin()).isTrue();
}
```

7.1.5 Testowanie mechanizmów bezpieczeństwa

Bezpieczeństwo aplikacji zostało poddane rygorystycznym procedurom weryfikacyjnym ukierunkowanym na kluczowe aspekty ochrony danych oraz kontroli dostępu. Procedura testowa obejmowała kompleksową analizę generowania i walidacji tokenów JWT, weryfikując ich integralność strukturalną oraz poprawność zawartych informacji uwierzytelniających. Przeprowadzono również testy mechanizmów filtrowania żądań HTTP, koncentrując się na prawidłowości interpretacji nagłówków autoryzacyjnych w kontekście zabezpieczenia interfejsu API. Dodatkowo wykonano weryfikację systemu uprawnień, potwierdzając precyzyjną kontrolę dostępu do funkcjonalności zgodnie z przydzielonymi rolami użytkowników. Procedura testowa objęła także systematyczną analizę obsługi scenariuszy wyjątkowych związanych z próbami nieuprawnionego dostępu oraz manipulacją danymi uwierzytelniającymi. Otrzymane rezultaty potwierdziły niezawodność zaimplementowanych rozwiązań zabezpieczających, jednocześnie identyfikując potencjalne obszary optymalizacji w kolejnych wersjach systemu.


```

@Test
@DisplayName("Generowanie tokenu JWT dla użytkownika")
void generateToken_shouldCreateValidToken() {
    // When
    String token = jwtUtil.generateToken(USERNAME);

    // Then
    assertNotNull(token);
    assertEquals(USERNAME, jwtUtil.extractUsername(token));
}

```

```

@Test
@DisplayName("Filtr powinien przetworzyć żądanie z prawidłowym tokenem JWT")
void shouldProcessRequestWithValidJwtToken()
    throws ServletException, IOException {
    // Given
    String token = "validToken";
    String username = "testUser";

    when(request.getHeader(HttpHeaders.AUTHORIZATION))
        .thenReturn("Bearer " + token);
    when(jwtUtil.extractUsername(token))
        .thenReturn(username);
    when(userDetailsService.loadUserByUsername(username))
        .thenReturn(userDetails);
    when(jwtUtil.validateToken(token, userDetails.getUsername()))
        .thenReturn(true);

    // When
    ReflectionTestUtils.invokeMethod(
        jwtAuthenticationFilter, "doFilterInternal",
        request, response, filterChain
    );

    // Then
    verify(filterChain).doFilter(request, response);
    verify(userDetailsService).loadUserByUsername(username);
    verify(jwtUtil).validateToken(token, userDetails.getUsername());
}

```

7.1.6 Testowanie obsługi wyjątków

Architektura aplikacji obejmuje rozbudowany system obsługi wyjątków, który został poddany rygorystycznym testom weryfikacyjnym. Proces testowania koncentrował się na poprawności mapowania wyjątków na odpowiednie kody odpowiedzi HTTP oraz generowaniu czytelnych komunikatów diagnostycznych. Przeprowadzone testy objęły weryfikację zachowania łańcucha odpowiedzialności w przetwarzaniu wyjątków, zapewniając właściwą delegację obsługi do odpowiednich komponentów systemu. Dodatkowo zweryfikowano reakcje systemu na specyficzne przypadki błędów, które mogłyby potencjalnie

zagrozić stabilności aplikacji. Wyniki testów potwierdziły skuteczność implementacji mechanizmu obsługi wyjątków, jednocześnie identyfikując możliwe obszary optymalizacji w kontekście obsługi sytuacji krytycznych.

```
@Test
@DisplayName("handle powinien zwrócić odpowiedź HTTP 404 dla
ResourceNotFoundException")
void handle_shouldReturnNotFoundResponseForResourceNotFoundException() {
    // Given
    ResourceNotFoundException exception =
        new ResourceNotFoundException("Resource not found");

    // When
    ResponseEntity<?> response = handler.handle(exception);

    // Then
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
    assertThat(response.getBody()).assertInstanceOf(ErrorResponse.class);
    ErrorResponse errorResponse = (ErrorResponse) response.getBody();
    assertThat(errorResponse.getMessage()).isEqualTo("Resource not found");
}
```

7.1.7 Wykryte i naprawione błędy

Implementacja kompleksowej metodyki testowania umożliwiła identyfikację i eliminację szeregu potencjalnych problemów, które mogłyby negatywnie wpłynąć na funkcjonalność oraz bezpieczeństwo systemu.

W zakresie walidacji danych wejściowych wykryto istotne niedostatki związane z weryfikacją pól encji, szczególnie w kontekście niepustych referencji. Zidentyfikowano również brak odpowiedniej obsługi przypadków brzegowych dla łańcuchów znakowych, obejmujących zarówno ich długość, jak i format. Dodatkowym obszarem wymagającym interwencji okazało się parsowanie dat w różnorodnych formatach, które w początkowej fazie wykazywało nieprawidłowości prowadzące do niespójności danych.

Proces testowania logiki biznesowej ujawnił nieprawidłowe zachowanie systemu podczas operacji archiwizacji zgłoszeń, gdzie nie uwzględniono kaskadowej archiwizacji wiadomości powiązanych ze zgłoszeniami. Ponadto, zidentyfikowano problemy w mechanizmie zarządzania statusem użytkowników, obejmujące operacje aktywacji i dezaktywacji, a także niepoprawne zachowanie podczas resetowania haseł użytkowników.

W krytycznym obszarze bezpieczeństwa systemu, testy umożliwiły wykrycie i naprawę luk w weryfikacji tokenów JWT, niedostateczną obsługę wygasłych

tokenów oraz niekompletne sprawdzanie uprawnień w określonych punktach końcowych API.

Konsekwentne stosowanie testów jednostkowych pozwoliło na wczesną identyfikację tych problemów, co znacząco wpłynęło na stabilność i niezawodność finalnej wersji systemu.

7.2 Testy integracyjne

Przedstawienie procesu testowania interakcji między komponentami systemu, ze szczególnym uwzględnieniem testów REST API. Opis wykorzystania narzędzi takich jak Spring Test i TestRestTemplate. Omówienie wykrytych problemów integracyjnych i sposobów ich rozwiązania.

7.3 Testy akceptacyjne

Opis scenariuszy testowych z perspektywy użytkownika końcowego, metodologia przeprowadzania testów, wykorzystane narzędzia. Prezentacja wybranych przypadków użycia i uzyskanych rezultatów.

7.4 Testy responsywności

Omówienie testowania interfejsu użytkownika pod kątem różnych urządzeń i rozdzielczości. Przedstawienie metodologii testowania responsywności i wykorzystanych narzędzi. Przykłady wykrytych problemów z responsywnością i ich rozwiązania.

7.5 Podsumowanie procesu testowania

Analiza i statystyki przeprowadzonych testów, główne wnioski, zidentyfikowane obszary ryzyka oraz sposoby ich minimalizacji. Omówienie procesu naprawy i potwierdzania naprawy znalezionych błędów.

8 Przykładowy scenariusz wykorzystania systemu

Niniejszy rozdział prezentuje praktyczne aspekty funkcjonowania systemu wspomagającego komunikację między studentami a pracownikami dziekanatu. Celem prezentacji jest zobrazowanie kluczowych scenariuszy użycia systemu, które ilustrują jego praktyczną wartość i innowacyjne rozwiązania w procesie wymiany informacji.

8.1 Złożenie wniosku przez studenta

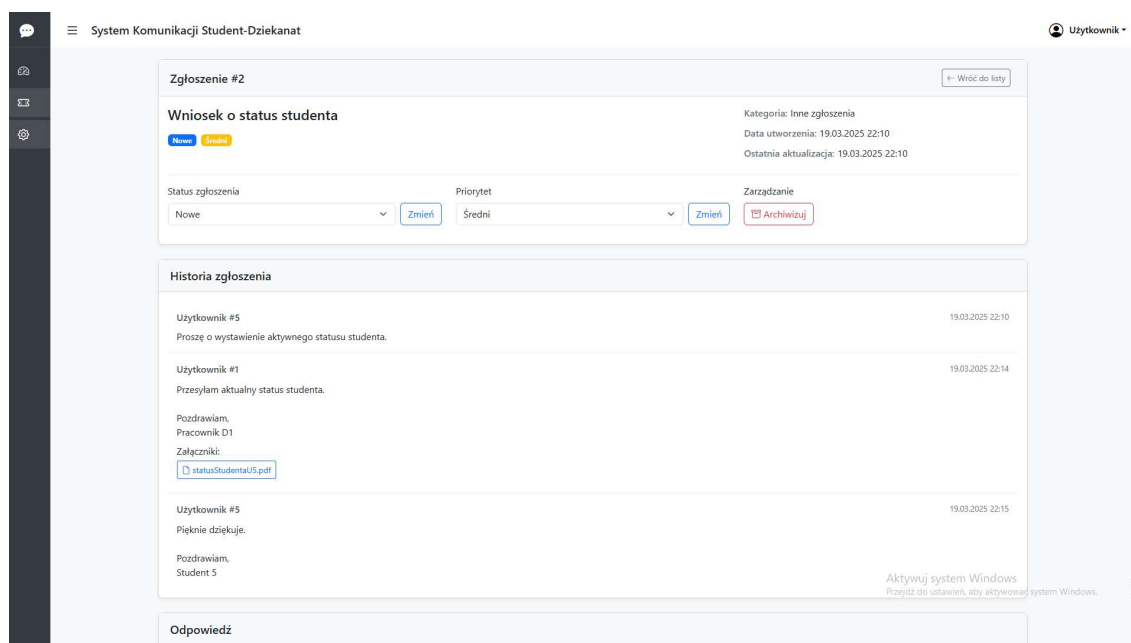
Student chcący złożyć wniosek o przeniesienie z trybu studiów stacjonarnych na niestacjonarne inicjuje proces poprzez utworzenie nowego zgłoszenia w systemie. Po zalogowaniu się na platformę, student wybiera opcję "Nowe zgłoszenie" i wypełnia formularz, określając kategorię wniosku, dodając szczegółowy opis oraz opcjonalnie dołączając wymagane dokumenty. System automatycznie generuje unikalny identyfikator zgłoszenia, umożliwiając studentowi śledzenie postępu rozpatrzenia wniosku. Pracownik dziekanatu zostaje powiadomiony o nowym zgłoszeniu i może przystąpić do jego weryfikacji.

The screenshot shows a web application interface for a student-dean communication system. On the left is a dark sidebar with icons for chat, home, mail, and settings. The main header includes the title 'System Komunikacji Student-Dziekanat', a button to toggle full-screen mode ('ABY ZAMKNAĆ TRYB PEŁNOKRANOWY, NACISNIJ F11'), and a user profile icon labeled 'Użytkownik'. The central focus is a 'Nowe zgłoszenie' (New Report) form. This form contains several input fields: 'Tytuł zgłoszenia' (Report Title), 'Kategoria' (Category) with a dropdown menu currently showing 'Wybierz kategorię', and 'Treść zgłoszenia' (Report Content) with a large text area. Below these is a 'Załącznik' (Attachment) section with a file selection button ('Wybierz plik') and a status indicator ('Nie wybrano pliku'). A note specifies allowed formats: 'Dozwolone formaty: PDF, JPG, PNG'. At the bottom right of the form are two buttons: 'Anuluj' (Cancel) and 'Utwórz zgłoszenie' (Create Report). A 'Pamiętaj' (Remember) checkbox is located at the top right of the form. In the bottom right corner of the application window, there is a Windows activation watermark: 'Aktywuj system Windows. Przejdź do ustawień, aby aktywować system Windows.'

Rysunek 20 Formularz składania nowego wniosku przez studenta

8.2 Komunikacja w ramach zgłoszenia

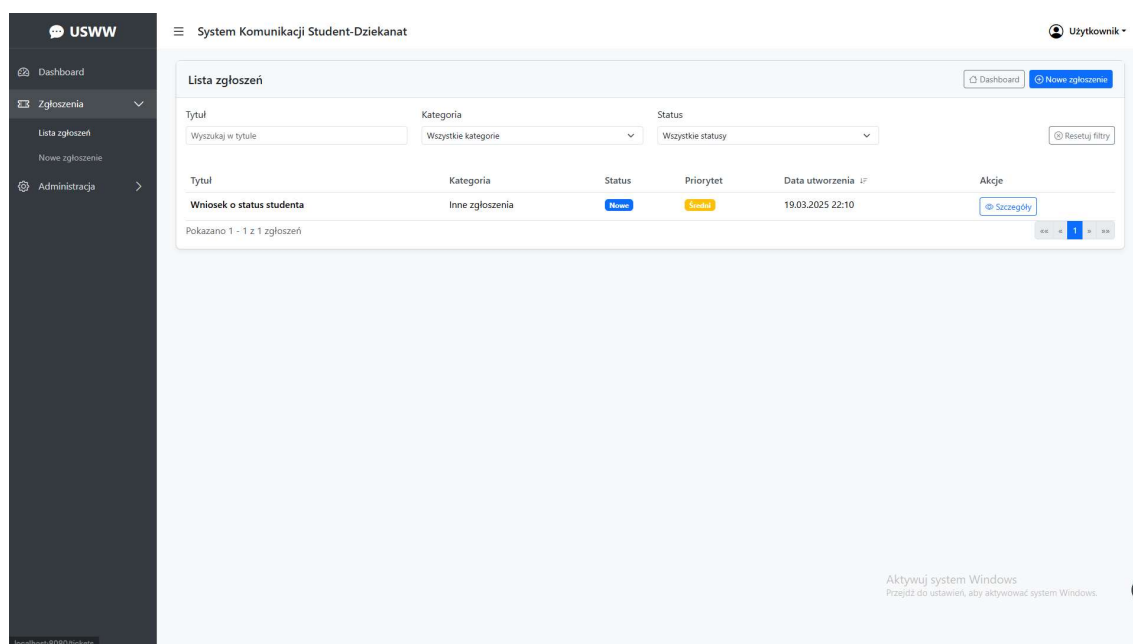
Po złożeniu wniosku, student może prowadzić korespondencję z pracownikiem dziekanatu bezpośrednio w kontekście danego zgłoszenia. System umożliwia dodawanie komentarzy, wymianę plików oraz śledzenie historii komunikacji. Pracownik dziekanatu może żądać dodatkowych informacji, proponować spotkanie lub wydać ostateczną decyzję. Wszystkie interakcje są rejestrowane i archiwizowane, zapewniając pełną transparentność procesu.



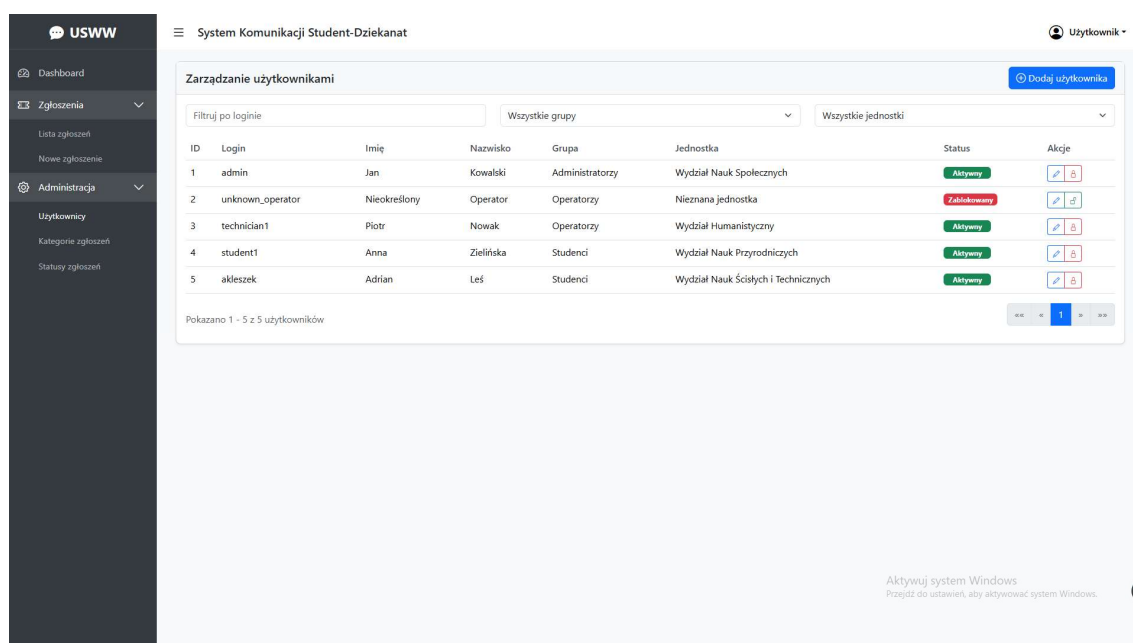
Rysunek 21 Widok komunikacji w ramach zgłoszenia z perspektywy pracownika dziekanatu

8.3 Zarządzanie zgłoszeniami przez administratora

Administrator systemu posiada rozszerzone uprawnienia umożliwiające kompleksowe zarządzanie użytkownikami i zgłoszeniami. Może tworzyć konta użytkowników, definiować ich role, przypisywać uprawnienia oraz monitorować aktywność w systemie. Dodatkowo administrator może przeprowadzać archiwizację starych zgłoszeń, zarządzać kategoriami zgłoszeń oraz kontrolować dostęp do systemu.



Rysunek 22 Panel listy zgłoszeń



Rysunek 23 Panel administracyjny zarządzania użytkownikami

Przedstawione scenariusze demonstrują kluczowe możliwości systemu, podkreślając jego użyteczność w usprawnieniu komunikacji między studentami a pracownikami dziekanatu. System nie tylko ułatwia składanie wniosków, ale również zapewnia przejrzystość, dostępność i efektywność procesów administracyjnych.

9 Zakończenie

Celem pracy było zaprojektowanie i implementacja systemu wspomagającego proces komunikacji między studentami a pracownikami dziekanatu, który umożliwia efektywną, bezkontaktową wymianę informacji, dokumentów i wniosków drogą elektroniczną.

Zrealizowany system oferuje kompleksowe rozwiązanie komunikacyjne oparte na mechanizmie zgłoszeń. Kluczowe funkcjonalności obejmują tworzenie nowych zgłoszeń, przeglądanie aktywnych i archiwalnych spraw, filtrowanie zgłoszeń, wymianę wiadomości oraz dołączanie załączników. System nie tylko usprawnia komunikację, ale również służy jako elektroniczne archiwum dokumentów związanych z realizacją zgłoszeń.

Podczas realizacji projektu wykorzystano nowoczesne technologie, które zapewniły wysoką jakość i wydajność rozwiązania:

- Warstwa serwerowa: Java 17 z szablonem Spring Boot 3.2.2
- Klient: Angular 17
- Baza danych: Microsoft SQL Server 2019
- Narzędzie budowania: Gradle 7.5

Proces implementacji systemu dostarczył wielu cennych doświadczeń i spostrzeżeń. Głównym wyzwaniem było zaprojektowanie intuicyjnego interfejsu użytkownika, który jednocześnie zapewnia kompleksową funkcjonalność. Szczególną uwagę poświęcono bezpieczeństwu danych, implementując mechanizmy autoryzacji i uwierzytelniania oparte na JSON Web Tokens.

Zastosowane podejście architektoniczne oparte na wzorcu REST API oraz trójwarstwowa architektura systemu pozwoliły na stworzenie skalowalnego i łatwego w utrzymaniu rozwiązania. Podział na moduły w warstwie prezentacji i serwerowej ułatwia przyszłe rozszerzenia funkcjonalności.

Kierunki dalszego rozwoju systemu obejmują:

- Implementację zaawansowanych mechanizmów raportowania i analityki zgłoszeń,
- Rozbudowę systemu powiadomień o dodatkowe kanały komunikacji,
- Integrację z systemami uczelnianymi,
- Rozszerzenie funkcjonalności o zaawansowane zarządzanie dokumentami.

Projekt stanowi kompleksowe rozwiązanie komunikacyjne, które nie tylko usprawnia procesy administracyjne, ale również znacząco podnosi komfort

obsługi studentów. Zastosowane nowoczesne technologie oraz przemyślana architektura systemu stwarzają solidne fundamenty dla dalszego rozwoju i doskonalenia rozwiązania.

Realizacja projektu była cennym doświadczeniem zawodowym, pozwalającym na praktyczne zastosowanie wiedzy z zakresu inżynierii oprogramowania, projektowania systemów informatycznych oraz nowoczesnych technologii webowych.

10 Bibliografia

1. Autor anonimowy, Przeniesienie (studia stacjonarne i niestacjonarne), <https://us.edu.pl/wydzial/wnst/studia/student/regulaminy/>, [dostępne: 15.02.2023].
2. Autor anonimowy, O systemie, <https://www.gov.pl/web/ezd-rp/o-ezd-rp>, [dostępne: 25.02.2024]
3. Autor anonimowy, Zendesk for Education, <https://www.zendesk.com/education/>, [dostępne: 25.02.2024]
4. Autor anonimowy, Your guide to Slack for higher education, <https://slack.com/resources/using-slack/your-guide-to-slack-for-higher-education>, [dostępne: 25.02.2024]
5. K. Wiegers, C. Hokanson , „Specyfikacja wymagań oprogramowania. Kluczowe praktyki analizy biznesowej”, Helion, 2024
6. Pressman, R. S., Architectural design, "Software Engineering: A Practitioner's Approach.", McGraw-Hill Education, 2014
7. Autor anonimowy, Spring Boot, <https://spring.io/projects/spring-boot>, [dostępne: 15.03.2024]
8. Autor anonimowy, Using Angular routes in a single-page application, <https://angular.io/guide/router-tutorial>, [dostępne: 15.03.2024]
9. Randolph West, Mike Ray, What is SQL Server?, <https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16>, [dostępne: 15.03.2024]
10. R. C. Martin, „Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów”, Helion, 2022
11. E. Gamma, R. Helm, R. Johnson, J. Vlissides, „Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku”, Helion, 2021

11 Spis rysunków

Rysunek 1 Aktualny obieg dokumentów w dziekanacie studenckim.....	9
Rysunek 2 Wniosek o przeniesienie z formy studiów stacjonarnych na niestacjonarne.....	10
Rysunek 3 Moduł podań systemu USOS UŚ	14
Rysunek 4 Składanie nowego podania w systemie USOS UŚ z listy określonych podań.....	14
Rysunek 5 Ekran startowy systemu ERP RP	16
Rysunek 6 Ekran główny platformy Zendesk.....	17
Rysunek 7 Interfejs platformy Slack.....	18
Rysunek 8 Schemat architektury REST API	29
Rysunek 9 Model relacyjnej bazy danych systemu wspierającego komunikację pomiędzy studentem a pracownikiem dziekanatu	32
Rysunek 10 Projekt ekranu logowania do systemu.....	34
Rysunek 11 Projekt ekranu głównego systemu dla studenta i pracownika dziekanatu.....	34
Rysunek 12 Projekt ekranu głównego systemu dla administratora systemu	35
Rysunek 13 Formularz utworzenia nowego zgłoszenia	35
Rysunek 14 Formatka dla podglądu aktywnych i zamkniętych zgłoszeń	36
Rysunek 15 Formularz podglądu i wysłania wiadomości w ramach istniejącego zgłoszenia	36
Rysunek 16 Formularz podglądu zamkniętego lub zarchiwizowanego zgłoszenia	37
Rysunek 17 Widok menu dostępnego dla administratora systemu	37
Rysunek 18 Struktura projektu warstwy logiki biznesowej	40
Rysunek 19 Struktura projektu warstwy prezentacji	48
Rysunek 20 Formularz składania nowego wniosku przez studenta	60
Rysunek 21 Widok komunikacji w ramach zgłoszenia z perspektywy pracownika dziekanatu	61
Rysunek 22 Panel listy zgłoszeń	62
Rysunek 23 Panel administracyjny zarządzania użytkownikami	62

12 Spis tabel

Tabela 1 Wymagania funkcjonalne.....	23
--------------------------------------	----