



AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

Zdalna szkoła - aplikacja webowa

Remote school- web application

Autor:

Daniel Kurosz

Kierunek studiów:

Informatyka

Opiekun pracy:

dr inż. Grzegorz Rogus

Kraków, 2021

Spis treści

| | |
|--|----|
| 1. Wstęp | 5 |
| 2. Systemy LMS | 7 |
| 2.1. Definicja i rozwój systemów LMS | 7 |
| 2.2. Uzasadnienie wyboru pracy | 8 |
| 2.3. Przegląd dostępnych platform | 8 |
| 2.3.1. Moodle | 8 |
| 2.3.2. Blackboard LMS | 9 |
| 2.3.3. Canvas | 9 |
| 3. Mikroserwisy | 11 |
| 3.1. Cechy mikroserwisów | 11 |
| 3.2. Zalety i wady stosowania architektury mikroserwisów | 12 |
| 4. Analiza wymagań | 13 |
| 4.1. Cel pracy | 13 |
| 4.2. Wymagania | 14 |
| 4.2.1. Wymagania funkcjonalne | 14 |
| 4.2.2. Wymagania нефункционалне | 15 |
| 4.3. Identyfikacja aktorów | 16 |
| 4.4. Funkcje systemu | 17 |
| 4.4.1. Diagramy przypadków użycia | 17 |
| 4.4.2. Scenariusze przypadków użycia | 19 |
| 5. Projekt architektury | 25 |
| 5.1. Architektura | 25 |
| 5.1.1. Podział według kontekstów | 25 |
| 5.1.2. Elementy systemu | 26 |
| 5.1.3. Sposób komunikacji między serwisami | 28 |

| | |
|--|-----------|
| 5.2. Model danych | 29 |
| 5.3. Autoryzacja i uwierzytelnianie..... | 33 |
| 6. Implementacja..... | 37 |
| 6.1. Rejestracja, autoryzacja i uwierzytelnianie | 37 |
| 6.2. API Gateway | 40 |
| 6.3. Broker wiadomości..... | 40 |
| 6.3.1. Wysłanie wiadomości | 40 |
| 6.3.2. Odebranie wiadomości..... | 41 |
| 6.4. Komunikacja z bazą danych | 42 |
| 6.5. Struktura serwisów | 43 |
| 7. Wykorzystane technologie..... | 45 |
| 7.1. Języki programowania..... | 45 |
| 7.2. Oprogramowanie, biblioteki i frameworki | 45 |
| 7.3. Narzędzia..... | 46 |
| 8. Interfejs systemu | 47 |
| 9. Testy..... | 51 |
| 10. Wdrożenie aplikacji | 53 |
| 11. Podsumowanie..... | 55 |
| 11.1. Podsumowanie pracy | 55 |
| 11.2. Możliwy rozwój | 55 |

1. Wstęp

W dzisiejszych czasach technologia pomaga nam w praktycznie każdym aspekcie życia. Nie inaczej jest z edukacją. Z pomocą przychodzą systemy LMS (Learning Management System), których zadaniem jest dostarczanie niezbędnych narzędzi do zdalnej edukacji. Do głównych zadań takich systemów należy tworzenie i zarządzanie kursami, udostępnianie materiałów naukowych, tworzenie egzaminów i zadań. Większość systemów umożliwia również sporządzanie raportów i statystyk z kursów, dzięki którym nauczyciel może efektywniej przygotowywać materiały dydaktyczne.

Rok 2020 i pandemia covid-19 sprawiła, że systemy, które wcześniej były często tylko dodatkiem i uzupełnieniem do tradycyjnego sposobu nauczania, stały się niezbędnym narzędziem dla uczniów i nauczycieli z całego świata. Szkoły i uczelnie musiały z dnia na dzień zaadaptować się do nowych warunków, szukając odpowiednich narzędzi do dostarczania edukacji na wysokim poziomie, zachowując przy tym prostotę użytkowania.

Celem niniejszej pracy jest zaprojektowanie systemu do zdalnego nauczania w oparciu o architekturę mikroserwisów z wykorzystaniem nowoczesnych technologii, a także implementacje prototypu. Projekt zakłada zaprojektowanie modułów dostarczających niezbędne funkcjonalności do tworzenia kursów w internecie. System posiadać będzie przyjazny w obsłudze interfejs webowy, dzięki czemu użytkownik nie będzie musiał instalować oprogramowania na swoim komputerze.

Praca składa się z części teoretycznej, oraz praktycznej. W pierwszej części omówiono systemy LMS, rozwój branży na rynku, wraz z prognozą na najbliższe lata. Przedstawione zostały najpopularniejsze rozwiązania, ich plusy, minusy. Przeanalizowane zostanie co użytkownikom takich systemów się podoba, a co niekoniecznie. Kolejnym tematem omówionym w pracy są mikroserwisy. Przedstawione zostały główne zasady budowania systemu w oparciu o mikro-usługi, zalety oraz wady takiego rozwiązania.

W dalszej części pracy została przeprowadzona analiza wymagań. Określony został cel pracy, wymagania funkcjonalne jak i нефункционалне. Omówieni zostali aktorzy projektowanego systemu, a także przedstawione zostały diagramy i scenariusze przypadków użycia.

W kolejnym rozdziale skupiono się na projekcie systemu. Do głównych zadań jakie należało wykonać było podzielenie systemu według spójnych kontekstów. Należało również zaprojektować wszystkie bazy danych obecne w systemie. W tym fragmencie omówiono również sposób zabezpieczenia mikroservisów.

Następnym etapem pracy jest implementacja systemu. W tym rozdziale przedstawione są fragmenty projektowanego kodu, dzięki którym można zobaczyć w jaki sposób zaimplementowane zostały problemy napotkane podczas tworzenia projektu.

Kolejny rozdział dotyczy wykorzystanych technologii podczas implementacji systemu. Krótko omówione zostały języki programowania, biblioteki, frameworki, systemy i narzędzia, bez których napisanie projektu byłoby znacznie utrudnione.

Ostatnimi trzema praktycznymi fragmentami pracy są rozdziały dotyczące interfejsu aplikacji, testów i możliwości wdrożenia. W rozdziale omawiającym interfejs przedstawione zostały zrzuty ekranu aplikacji internetowej. W kolejnym rozdziale omówiono rodzaje testów, którymi sprawdzano jakość pisanego oprogramowania. Ostatnim praktycznym rozdziałem jest opis wdrożenia systemu. W tym miejscu przedstawiona została platforma Docker i jej możliwości tworzenia kontenerów.

Ostatnią częścią pracy jest podsumowanie wykonanych prac. Wykonany projekt systemu umożliwia na utworzenie niezbędnych funkcjonalności do obsługi kursów w sposób zdalny. Zaimplementowany prototyp dostarcza solidny szkielet do rozbudowy narzędzia, dzięki czemu w wolnych chwilach system będzie mógł być rozwijany. Mimo, że na rynku systemów do zdalnego nauczania jest kilku dużych graczy, sam rynek nieustannie rośnie i ewoluuje, dzięki czemu jest możliwość znalezienia własnego pola do rozwijania autorskiego narzędzia.

2. Systemy LMS

2.1. Definicja i rozwój systemów LMS

Systemy LMS wspomagają szkoły, uczelnie, a także firmy w zdalnej edukacji. Do głównych zadań aplikacji należy udostępnianie materiałów edukacyjnych, zarządzanie kursami oraz przeprowadzanie egzaminów. Umożliwiają nauczanie o każdej porze dnia, niezależnie od miejsca gdzie znajduje się student czy nauczyciel. Pozwalają na analize postępów w edukacji, przez co sam proces nauczania staje się efektywniejszy [1].

Branża LMS cały czas się rozwija i z roku na rok zwiększa się wartość rynkowa oraz ilość użytkowników. Według raportu sporządzonego przez Markets And Markets wartość sektora LMS w 2020 roku wynosi 13,4 miliarda \$, natomiast prognozowaną wartością na 2025 jest 25,7 miliarda \$, z wynikiem współczynnika CAGR (Compound Annual Growth Rate) na poziomie 14%. Jak wynika z wykresu 2.1, systemy do zdalnego nauczania największe znaczenie mają w Ameryce Północnej, natomiast prognozuje się szybko rosnące zapotrzebowanie na tego typu oprogramowanie w Europie i regionie APAC (Południowa Azja, Australia i Oceania) [2].



Rys. 2.1. Prognozowana wartość rynkowa branży LMS w latach 2018 - 2025

2.2. Uzasadnienie wyboru pracy

Systemy LMS zyskują coraz większe znaczenie w procesie nauczania. Akademia Górniczo-Hutnicza w Krakowie posiada własny system LMS, jakim jest platforma UPeL [3]. Oparta na oprogramowaniu Moodle pozwala na podstawowe operacje zarządzania kursami w internecie. Niestety, nie jest to platforma bez wad. Podczas wypełniania egzaminu strona znacząco zwalnia, a przy rozwiązywaniu pytań łatwo zaznaczyć przypadkowo złą odpowiedź. Również szata graficzna aplikacji wydaje się odbiegać od dzisiejszych standardów. Opierając się na opiniach o danej platformie można dojść do wniosku, że obecne rozwiązanie nie spełnia w pełni oczekiwań użytkowników. Przeglądając fora internetowe na temat innych systemów LMS, można zauważyć, że najczęściej wymienianymi wadami jest nieintuicyjność, oraz mało przyjazny interfejs. Przedstawiona analiza pozwala na stwierdzenie, że na rynku brakuje oprogramowania, które spełniać będzie wszystkie wymagania klientów.

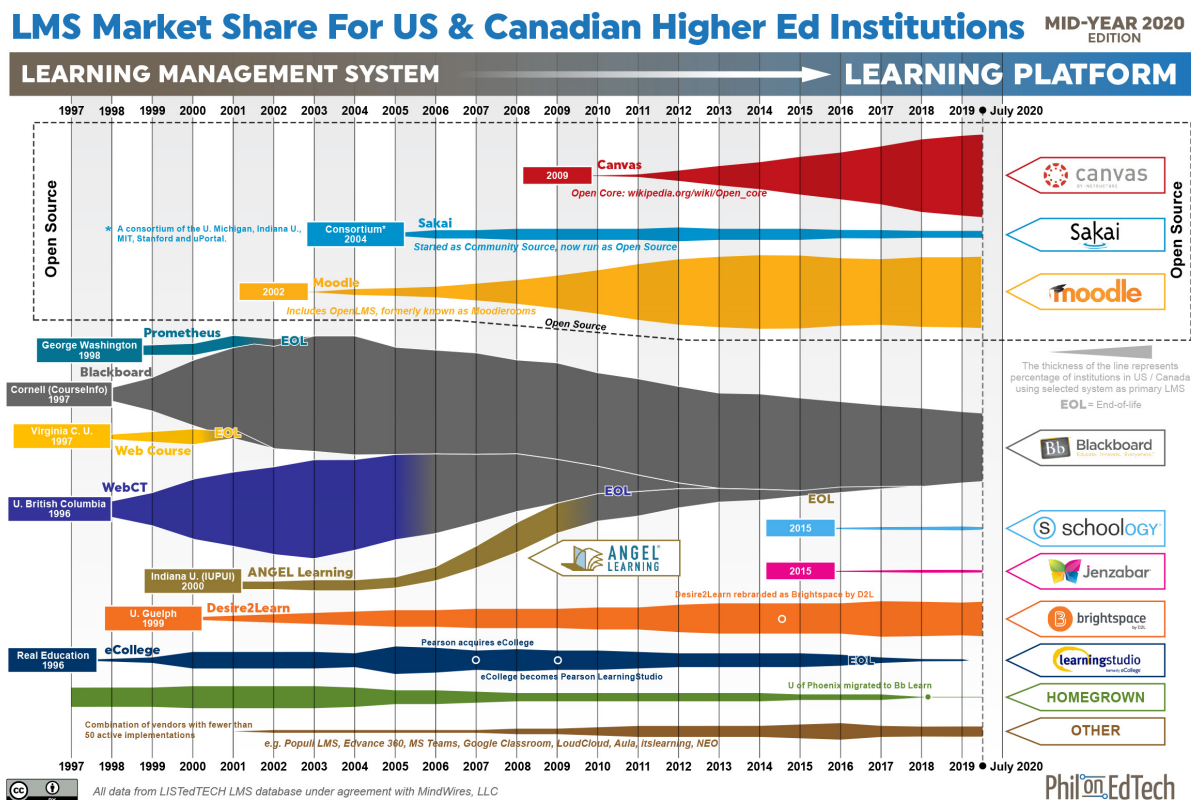
Dane z wykresu 2.1 pozwalają na stwierdzenie, że e-learning będzie rozwijał się w szybkim tempie, zyskując większe znaczenie w edukacji. Warto więc stworzyć własną aplikację, dołączając do tak prężnie rozwijającej się gałęzi branży, tworząc autorskie rozwiązanie.

2.3. Przegląd dostępnych platform

Na rynku systemów do zdalnej edukacji istnieje wiele rozwiązań. Są to zarówno systemy open-source jak i płatne aplikacje. Prym wiodą głównie 3 aplikacje: Moodle i Canvas jako przedstawiciele darmowego oprogramowania, a także płatny Blackboard LMS.

2.3.1. Moodle

Moodle to najpopularniejszy system open-source do zdalnego nauczania. Aplikacja działa w 249 krajach z 189 tysiącami zarejestrowanych stron i ilością użytkowników na poziomie 250 milionów [4]. Platforma przeznaczona jest dla szkół i uczelni wyższych oferując szeroki zakres zarządzania kursami. Wokół systemu urosła znaczna społeczność, dzięki czemu aplikacja jest cały czas wspierana i aktualizowana. Dużym plusem aplikacji jest pokaźna liczba pluginów, które pozwalają dostosować oprogramowanie do własnych potrzeb. Zaletą jest również wspomniana wyżej duża społeczność i bezpłatność użycia. Do najczęściej wymienianych problemów można zaliczyć nieprzejrzysty i przestarzały interfejs graficzny, oraz wysoki próg wejścia do zarządzania systemem. [5] Moodle napisany jest przy użyciu języka PHP.



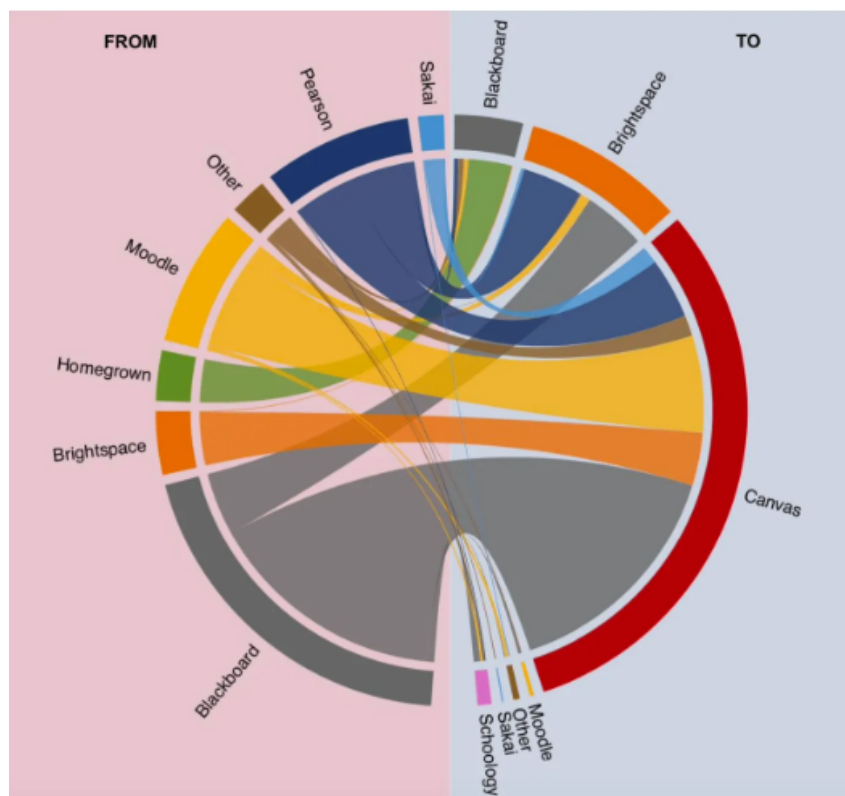
Rys. 2.2. Udział systemów LMS na rynku amerykańskim w latach 1997 - 2019

2.3.2. Blackboard LMS

Platnym przedstawicielem systemów do zdalnego nauczania jest Blackboard. Ceny platformy ustalane są indywidualnie z uczelniami i zależą od takich czynników jak ilość użytkowników czy zakres usług i włączonych modułów. System pozwala na organizowanie spotkań w wirtualnych salach wykładowych, tworzenie kursów i egzaminów, a także posiada szeroki zakres rozwiązań analitycznych [6]. Umożliwia również wysyłanie wiadomości i dyskusowanie pod każdym z kursów. Z roku na rok udziały aplikacji jednak spadają co spowodowane może być przestarzałym interfejsem i bardzo wysokimi opłatami licencyjnymi za system.

2.3.3. Canvas

Canvas to otwartoźródłowe oprogramowanie do nauczania zdalnego od firmy Instructure. Popularność platformy z roku na rok rośnie. Jak widać na wykresie 2.3, w latach 2018-2020 aplikacja zyskała użytkowników z wielu innych systemów LMS, głównie omawianych wyżej Blackboard i Moodle. Ponadto przypadków, kiedy użytkownik rezygnował z Canvas'a na rzecz innego oprogramowania było na tyle mało, że nie odnotowano ich na wykresie [7].



Rys. 2.3. Wykres migracji między systemami LMS w latach 2018 - 2020

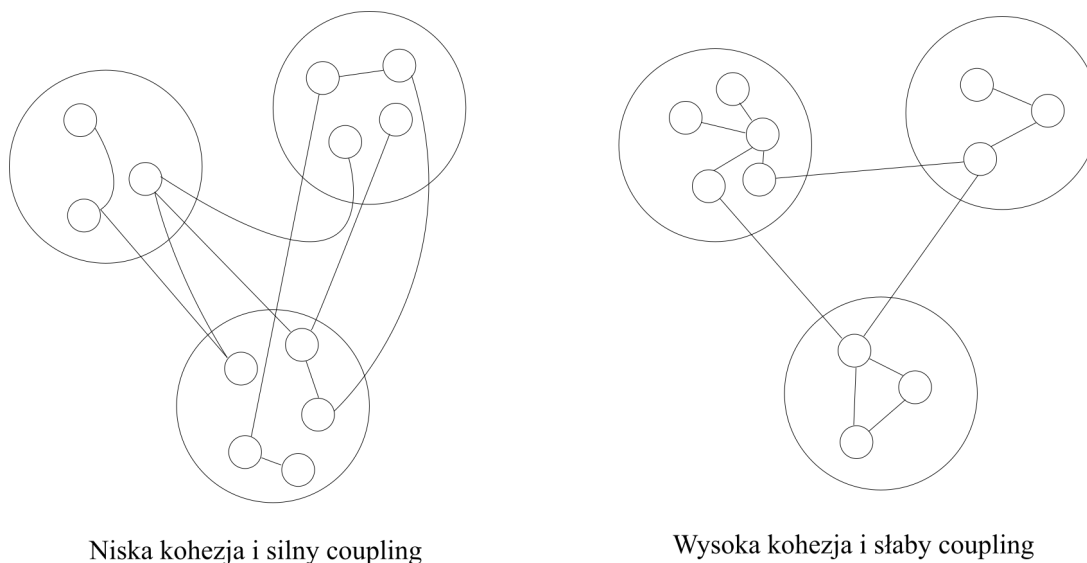
Platforma pisana jest w językach programowania Ruby i Javascript. Całość kodu źródłowego umieszczona jest w serwisie Github [8]. Zaletami aplikacji jest prostota użytkowania i przyjazny interfejs. Jako główne minusy, użytkownicy wskazują na problemy z przesyłaniem wiadomości i obsługą powiadomień, a także problematyczną nawigację w niektórych miejscach strony.

3. Mikroserwisy

Sposób wytwarzania oprogramowania nieustannie ewoluuje. W ostatnich latach popularność zdobywają mikroserwisy. Dobrze zaprojektowaną architekturę mikroservisów można rozpoznać po autonomiczności poszczególnych serwisów, wysokiej kohezji i słabym couplingu. W wydzielaniu mikroservisów i definiowaniu zależności nieodzowna jest znajomość DDD (Domain Driven Development) wraz z koncepcją bounded context, czyli podzieleniem aplikacji na spójne konteksty.

3.1. Cechy mikroservisów

1. Autonomiczność - Kluczową cechą mikroservisów jest jego autonomiczność. Serwisy nie mogą mieć współdzielonych baz danych, a komunikacja odbywa się przez sieć. Dzięki autonomiczności serwis może być rozwijany i wdrażany niezależnie od innych serwisów systemu. Każdy serwis można przypisać osobnej grupie deweloperów, ponadto mogą być pisane w różnych językach programowania, które najlepiej pasują do problemów rozwiązywanych przez dany serwis.
2. Wysoka kohezja - Kohezja oznacza poziom pasowania do siebie klas i funkcji w obrębie danego modułu czy serwisu. W przypadku mikroservisów należy dążyć do jak najwyższej kohezji. Dzięki wysokiej spójności zmieniając funkcjonalność w serwisie, zmieniamy ją tylko w tym jednym serwisie, co przyspiesza i ułatwia proces tworzenia oprogramowania. Aby uzyskać kohezję na odpowiednio wysokim poziomie kluczowe jest prawidłowe definiowanie kontekstów i granic między nimi.
3. Słaby coupling - W momencie kiedy między serwisami dochodzi do zbyt wielu zależności, proces modyfikowania i tworzenia nowych funkcjonalności znacznie się wydłuża oraz zwiększa się poziom skomplikowania. Projektując mikroserwisy należy zdefiniować konteksty i granice między nimi w taki sposób, aby połączeń między serwisami było tak mało jak to tylko możliwe [9].



Rys. 3.1. Porównanie systemu z niską kohezją i wieloma powiązaniem a systemem z wysoką kohezją i małą ilością zależności

Na rysunku 3.1 porównano dwa systemy. W pierwszym z nich podzielono w zły sposób konteksty, przez co między serwisami dochodzi do wielu powiązań, a same moduły wewnątrz serwisów nie mają ze sobą wiele wspólnego. Drugi system został znacznie lepiej podzielony, powiązań jest niewiele, a każdy serwis jest spójny. Diagram jest czytelny i bardzo szybko można zrozumieć zależności między poszczególnymi modułami aplikacji.

3.2. Zalety i wady stosowania architektury mikroserwisów

Głównymi zaletami omawianej architektury jest możliwość rozwijania serwisów niezależnie od siebie, w dopasowanych do danego problemu technologii. W przypadku nieprawidłowego działania aplikacji, dużo łatwiej jest znaleźć i naprawić przyczynę. Podział na mniejsze serwisy umożliwia również łatwiejsze wdrażanie nowych funkcjonalności. Wydajniejsza jest także skalowalność – dzięki mikroserwisom skalujemy tylko te serwisy które tego wymagają, a nie całą aplikację jak w przypadku monolitu.

Słabymi stronami mikroserwisów jest dodatkowa złożoność związana z komunikacją między serwisami. Problemem jest także zapewnienie transakcyjności danych – w przypadku monolitu jest to zapewnione na poziomie bazy danych, w architekturze mikroserwisów programista musi to zaimplementować w serwisie.

4. Analiza wymagań

Branża systemów LMS nieustannie się rozwija, zyskując coraz większe znaczenie w procesie edukacji. Niestety, obecne rozwiązania nie spełniają całkowicie oczekiwań konsumentów. Z przeprowadzonej analizy, wynika, że aplikacje często są mało intuicyjne i zbyt skomplikowane, co sprawia problemy niektórym wykładowcom, szczególnie tym starszym. Aplikacja powinna wspomagać nauczyciela w nauczaniu, a nie przeciążać mało przydatnymi funkcjami, czego następstwem może być frustracja i spadek jakości kształcenia. Zadaniem omawianej pracy jest stworzenie funkcjonalnej i intuicyjnej aplikacji rozwiązującej ten problem.

W niniejszym rozdziale określony zostanie cel pracy wraz z spisem wymagań funkcjonalnych jak i нефункциональных, które powinna spełniać aplikacja. Przedstawieni zostaną wszyscy aktorzy, omówiona zostanie ich rola w systemie. Ostatnim elementem rozdziału będą diagramy przypadków użycia, przedstawiające całościowe działanie aplikacji, wraz z scenariuszami będącymi dokładniejszą formą opisu funkcjonalności.

4.1. Cel pracy

Celem niniejszej pracy jest zaprojektowanie aplikacji webowej wspomagającej uczelnie w zdalnym nauczaniu, wraz z implementacją prototypu. Projekt powinien zawierać niezbędne narzędzia do przygotowywania kursów, zarządzania listą studentów oraz przeprowadzania egzaminów. Istotne jest również, aby aplikacja pozwalała na łatwą i szybką komunikację student-nauczyciel. Bardzo ważną cechą projektowanego rozwiązania jest prostota i intuicyjność użycia, tak aby każdy mógł bezproblemowo korzystać z platformy. Ideą pracy jest przygotowanie projektu w architekturze mikroservisów w taki sposób, aby aplikację można było w przystępny sposób rozwijać o nowe funkcjonalności.

4.2. Wymagania

Jedną z głównych zasad wytwarzania oprogramowania jest sprecyzowanie wymagań, które powinna spełniać aplikacja. Prawidłowo skonstruowany zestaw wymagań pomaga programistom zbudować system, który będzie robił dokładnie to, czego oczekuje klient. Ten etap wymaga dużej uwagi, ponieważ w przypadku, gdy określone zostaną niepełne lub błędne wymagania, tworzony system może okazać się bezużyteczny.

Definiowanie wymagań to proces płynny. W przypadku dużych systemów niemożliwym jest określenie wszystkich wymagań na samym starcie. Wraz z rozwojem aplikacji dochodzą nowe pomysły, zmieniają się technologie i sposoby rozwiązywania danego problemu. Aby system był konkurencyjny należy brać pod uwagę takie aspekty, ponieważ tylko ciągły rozwój jest w stanie zapewnić odpowiednią jakość wytwarzanej usługi. Należy przy tym uważać, aby nowe wymagania, były zgodne z tymi już istniejącymi [10].

4.2.1. Wymagania funkcjonalne

Wymienione poniżej wymagania powstały na skutek przeprowadzenia analizy dostępnych na rynku platform do zdalnego nauczania, oraz na podstawie własnego doświadczenia. Przeglądając różne fora internetowe, w których oceniane były omawiane wyżej systemy LMS (Moodle, Canvas oraz Blackboard) [5, 11], wyszczególnie zostały poniższe wymagania.

- Aplikacja powinna rozróżniać kilka ról użytkowników: dyrektor, nauczyciel i student. Ponadto funkcjonalność powinna zostać tak zaimplementowana, aby w łatwy sposób można było rozszerzyć aplikację o nowe role.
- Aplikacja powinna posiadać interfejs webowy działający na wszystkich popularnych przeglądarkach bez potrzeby instalacji dodatkowego oprogramowania.
- Nauczyciel w łatwy sposób powinien mieć możliwość przygotowywania lekcji. Stworzoną lekcję można udostępnić w publicznym repozytorium, skąd inni nauczyciele będą mogli ją pobrać do innych kursów, lub zostawić jako wersję prywatną.
- Aplikacja powinna dostarczyć moduł do tworzenia egzaminów. Pytania mogą być różnego typu. Początkowo aplikacja powinna posiadać przynajmniej 3 typy pytań do wyboru. W ramach jednego egzaminu można wymieszać rodzaje zadań.
- Egzamin powinien być dostępny do rozwiązania w określonym przez zarządzającego nim nauczyciela oknie czasowym. Po wysłaniu odpowiedzi system powinien ocenić wszystkie zadania zamknięte, a zadania otwarte wysłać do sprawdzenia przez nauczyciela.

- Pod każdą lekcją powinna być możliwość zostawiania komentarzy.
- Z każdego egzaminu nauczyciel powinien móc wygenerować raport podsumowujący test. W raporcie powinny znaleźć się takie informacje jak:
 - procentowa zdawalność testu,
 - ranking najlepiej i najgorzej rozwiązywanych pytań,
 - szczegółowe informacje rozwiązanych pytań poszczególnych studentów
- System powinien udostępniać interfejs dla dyrektora do zarządzania użytkownikami i właściwościami uczelni. Z poziomu panelu może zmieniać role użytkowników, dodawać nowych lub usuwać już istniejących.
- Aplikacja powinna dostarczyć odpowiednie narzędzie do wysyłania i odbierania wiadomości. Komunikacja aplikacji webowej z serwerem powinna być dwustronna, co zapewni odpowiedź zarządzanie powiadomieniami.
- Aplikacja powinna umożliwić wysyłanie i pobieranie materiałów w różnych formacie, m.in. pdf, obrazy czy pliki audio i wideo.

4.2.2. Wymagania niefunkcjonalne

- Intuicyjność – Użytkowanie systemu powinno być intuicyjne. Interfejs graficzny należy zaprojektować zgodnie z zasadami UI/UX.
- Elastyczność – Kod aplikacji powinien być elastyczny i łatwo modyfikowalny w celu dodawania nowych funkcjonalności oraz poprawiania już istniejących.
- Bezpieczeństwo – System powinien gwarantować bezpieczną rejestrację i logowanie użytkowników. Autoryzacja i autentykacja powinna przebiegać bez żadnych problemów, z wykorzystaniem nowoczesnych technik autoryzacyjnych.
- Niezawodność – System powinien być w stanie działać cały czas. Aplikacja musi gwarantować prawidłowe działanie kont użytkowników, czy zapisywać poprawnie odpowiedzi użytkowników na pytania egzaminacyjne.
- Nowoczesne technologie – System powinien być zaprojektowany przy użyciu nowoczesnych języków programowania. Aplikacja powinna być zaprojektowana zgodnie z najnowszymi technikami wytwarzania oprogramowania.

- **Czysty kod** – Oprogramowanie należy stworzyć zgodnie z ogólnie przyjętymi zasadami wytwarzania oprogramowania. Kod źródłowy powinien być czytelny i efektywny [12].
- **Wydajność** – System w fazie prototypowej powinien umożliwiać komunikację między kilkoma osobami w jednym czasie.
- **Internacjonalizacja** – Aplikacja webowa powinna wspierać różne języki, w początkowej wersji jest to język polski i angielski. Dodawanie nowych tłumaczeń powinno być proste i bez większych modyfikacji w systemie.

4.3. Identyfikacja aktorów

W projektowanym systemie wyszczególnione zostały trzy typy aktorów: osoby fizyczne, systemy oraz czas [13]. Niżej omówiony został każdy z nich.

1. Osoby fizyczne:

- **Gość** – Niezalogowany w systemie użytkownik.
- **Dyrektor** – Najważniejsza osoba w systemie. Dyrektor ma wgląd do każdego użytkownika w systemie, może nadawać im odpowiednie uprawnienia.
- **Nauczyciel** – Posiada uprawnienia do tworzenia lekcji i kursów. Zarządza i ocenia egzaminy oraz zadania. Otrzymuje także moduł do analizy postępów studentów w nauce.
- **Student** – Najliczniejsza grupa użytkowników w systemie. Student otrzymuje dostęp do kursów i lekcji tworzonych przez nauczycieli.

2. Systemy:

- **System uwierzytelniania** – Aplikacja w której przechowywane są loginy, hasła i inne podstawowe dane użytkowników. System umożliwia również bezpieczne odelegowanie uwierzytelniania do zewnętrznych aplikacji.
- **Aplikacje zewnętrzne** – Aplikacje implementujące standard OAuth, dzięki czemu możliwe jest potwierdzenie tożsamości logującego się użytkownika. Takimi aplikacjami są na przykład: Facebook, Github, Twitter.

3. Czas:

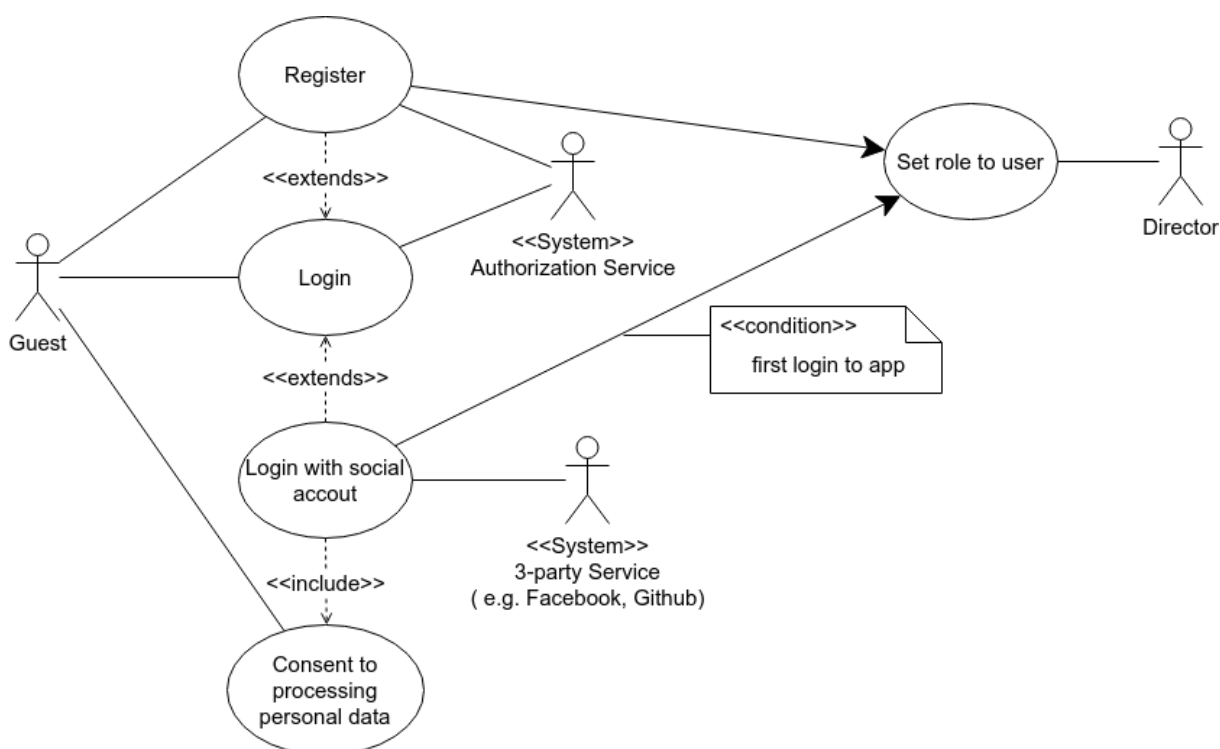
- **Termin wysłania** – Zdefiniowana przez nauczyciela data, do której uczeń powinien wysłać zadanie lub wypełnić egzamin.

4.4. Funkcje systemu

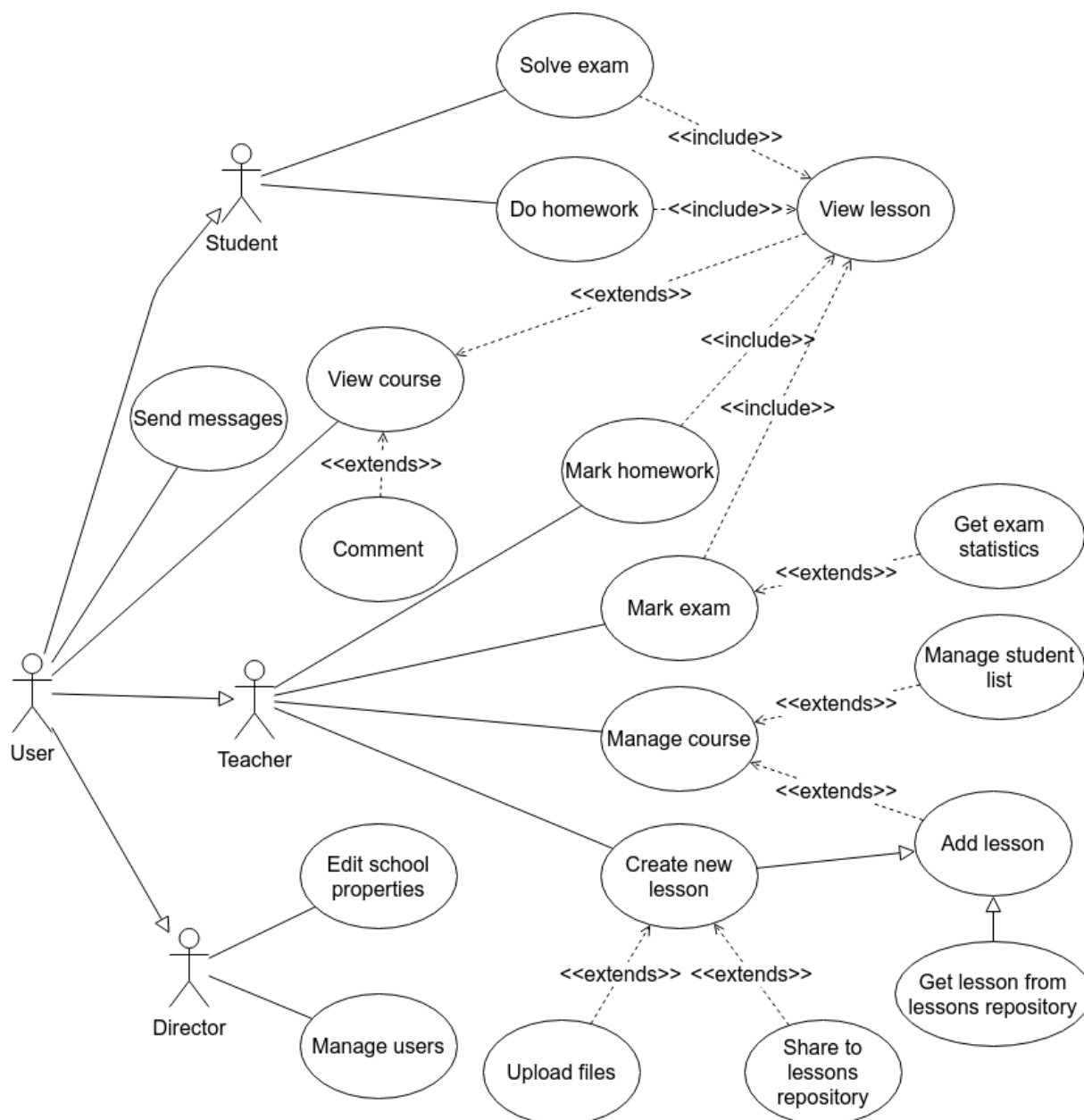
W niniejszym rozdziale przedstawione zostaną funkcjonalności projektowanego systemu. W pierwszym podrozdziale omówione zostaną diagramy przypadków użycia, wraz z krótkim opisem słownym. W kolejnym podrozdziale znajdują się scenariusze przypadków użycia. Pozwolą one na dokładniejsze zrozumienie projektowanych funkcjonalności.

4.4.1. Diagramy przypadków użycia

Na diagramie 4.1 przedstawiono sposoby na zalogowanie i rejestrację użytkowników. Logowanie może odbyć się albo przez podanie loginu i hasła, lub przy użyciu aplikacji zewnętrznej, która zapewnia uwierzytelnienie danej osoby. Taką funkcjonalność posiada większość aplikacji społeczności. Diagram 4.2 prezentuje ogólny obraz funkcjonalności aplikacji dla zalogowanych użytkowników. Każdy zalogowany użytkownik może wysyłać do innych wiadomości, oraz wyświetlać kursy. Nauczyciel może dodatkowo tworzyć nowe kursy, ma dostęp do repozytorium lekcji. Dyrektor jako jedyny w systemie może zarządzać uczestnikami, zmieniać im uprawnienia, a także usuwać ich z systemu.



Rys. 4.1. Diagram przedstawiający logowanie do aplikacji



Rys. 4.2. Diagram przypadków użycia dla zalogowanego użytkownika

4.4.2. Scenariusze przypadków użycia

| Nazwa | Rejestracja nowych użytkowników |
|-------------------------------|---|
| Numer | PU-1 |
| Aktorzy | Dyrektor, Gość, System uwierzytelniania, Aplikacja zewnętrzna |
| Opis | Gość chce utworzyć nowe konto w systemie |
| Warunki wstępne | Dyrektor ma utworzone konto |
| Główny przepływ zdarzeń | <ol style="list-style-type: none"> Gość wybiera opcję rejestracji Gość zostaje przeniesiony do Systemu uwierzytelniania Wybiera opcję standardowej rejestracji [A1: Rejestracja przy użyciu portalu zewnętrznego] Gość podaje potrzebne dane do rejestracji, zatwierdza [A2: Błędne dane rejestracji] <u>Zatwierdzenie rejestracji:</u> Dyrektor otrzymuje informację o rejestracji użytkownika Dyrektor ustawia zarejestrowanemu użytkownikowi odpowiednią rolę |
| Alternatywny przepływ zdarzeń | <p>A1: Rejestracja przy użyciu portalu zewnętrznego</p> <ol style="list-style-type: none"> Gość wybiera opcję zalogowania przy użyciu Aplikacji zewnętrznej. Gość zatwierdza zgodę o pobranie danych osobowym [A3: Brak zgody] Powrót do <u>Zatwierdzenie rejestracji</u> <p>A2: Błędne dane rejestracji</p> <ol style="list-style-type: none"> Gość proszony jest o ponowne wprowadzenie danych osobowych. W przypadku powodzenia, powrót do <u>Zatwierdzenie rejestracji</u> <p>A3: Brak zgody</p> <ol style="list-style-type: none"> Gość dostaje informację o braku dostępu do platformy |

Tabela 4.1. Scenariusz przypadku użycia: Rejestracja nowych użytkowników

| Nazwa | Tworzenie nowej lekcji |
|-------------------------------|--|
| Numer | PU-2 |
| Aktorzy | Nauczyciel |
| Opis | Nauczyciel tworzy nową lekcję |
| Warunki wstępne | Nauczyciel jest zalogowany w aplikacji Nauczyciel jest na stronie tworzenia lekcji |
| Główny przepływ zdarzeń | <ol style="list-style-type: none"> 1. Nauczyciel wybiera typ lekcji, jaką chce stworzyć: Wykład, Ćwiczenia albo Egzamin 2. Wyświetlony zostaje formularz z podstawowymi danymi na temat lekcji, między innymi: nazwa, opis 3. Nauczyciel przesyła dodatkowe materiały do lekcji 4. W przypadku lekcji typu 'Ćwiczenia', nauczyciel dodaje treść zadania, dla egzaminu tworzy pytania z odpowiedziami. 5. Nauczyciel ustawia zakres widoczności lekcji. W przypadku wybrania opcji prywatnej, dodaje kod dostępu 6. Nauczyciel potwierdza dodanie lekcji [<i>AI: Anulowanie tworzenia lekcji</i>] 7. Lekcja zostaje zapisana w repozytorium lekcji |
| Alternatywny przepływ zdarzeń | <i>AI: Anulowanie tworzenia lekcji</i> <ol style="list-style-type: none"> 1. Nauczyciel anuluje proces tworzenia lekcji 2. Z systemu usunięte zostają dodane wcześniej dane i przesłane materiały |

Tabela 4.2. Scenariusz przypadku użycia: Tworzenie nowej lekcji w systemie

| Nazwa | Tworzenie nowego kursu |
|--------------------------------------|---|
| Numer | PU-3 |
| Aktorzy | Nauczyciel |
| Opis | Nauczyciel tworzy nowy kurs |
| Warunki wstępne | Nauczyciel jest zalogowany w aplikacji Nauczyciel jest na stronie tworzenia kursu |
| Główny przepływ zdarzeń | <ol style="list-style-type: none"> 1. Nauczyciel wypełnia formularz z podstawowymi danymi na temat kursu, między innymi: nazwa, opis 2. Nauczyciel tworzy nową lekcję dla kursu (Patrz: PU-2) [<i>A1: Pobranie lekcji z repozytorium</i>] 3. Utworzona lekcja może zostać zmodyfikowana na potrzeby danego kursu. Nauczyciel może zmienić jej nazwę, opis, dodać skalę ocen i datę dostępności. 4. Nauczyciel ponawia kroki 2-3 w celu uzyskania odpowiedniej ilości lekcji w kursie. 5. Nauczyciel ustawia zakres widoczności kursu. W przypadku wybrania opcji prywatnej, dodaje kod dostępu 6. Nauczyciel potwierdza dodanie kursu [<i>A2: Anulowanie tworzenia kursu</i>] 7. Kurs zostaje zapisany w bazie danych |
| Alternatywny przepływ zdarzeń | <p><i>A1: Pobranie lekcji z repozytorium</i></p> <ol style="list-style-type: none"> 1. Nauczyciel zostaje przeniesiony do repozytorium dostępnych lekcji 2. Po znalezieniu lekcji, nauczyciel pobiera ją do swojego kursu 3. Powrót do kroku numer 3 głównego przepływu zdarzeń <p><i>[A2: Anulowanie tworzenia kursu]</i></p> <ol style="list-style-type: none"> 1. Nauczyciel anuluje proces tworzenia kursu |

Tabela 4.3. Scenariusz przypadku użycia: Tworzenie kursu w systemie

| Nazwa | Przesyłanie zadań |
|-------------------------------|---|
| Numer | PU-4 |
| Aktorzy | Uczeń, Nauczyciel, Termin wysłania |
| Opis | Uczeń przesyła zadanie |
| Warunki wstępne | Nauczyciel stworzył lekcję typu "Ćwiczenie" Uczeń ma dostęp do kursu |
| Główny przepływ zdarzeń | <p>1. Uczeń przechodzi do ćwiczenia [A3: Brak oddania zadania]</p> <p>2. Uczeń przesyła materiały [A1: Termin wysłania minął]</p> <p><u>Ocena zadania:</u></p> <p>3. Nauczyciel pobiera materiały wysłane przez ucznia</p> <p>4. Nauczyciel ocenia ćwiczenie użytkownika</p> |
| Alternatywny przepływ zdarzeń | <p>[A1: Termin wysłania minął]</p> <p>1. Termin wysłania zadania minął</p> <p>2. System wyświetla uczniowi o tym, że czas na oddanie zadania już minął</p> <p>3. Jeżeli nauczyciel ustawił w opcjach lekcji możliwość oddania zadania po czasie, uczeń przesyła materiały [A2: Termin oddania minął, blokada]</p> <p>4. Przejście do <u>Ocena zadania</u></p> <p>[A2: Termin oddania minął, blokada]</p> <p>1. System wyświetla uczniowi informację o braku możliwości oddania zadania</p> <p>2. Przejście do [A3: Brak oddania zadania]</p> <p>[A3: Brak oddania zadania]</p> <p>1. Uczeń nie wysłał zadania</p> <p>2. System wystawia uczniowi 0 punktów</p> |

Tabela 4.4. Scenariusz przypadku użycia: Przesyłanie zadań

| Nazwa | Przeprowadzanie egzaminu |
|-------------------------------|---|
| Numer | PU-5 |
| Aktorzy | Uczeń, Nauczyciel, Termin wysłania |
| Opis | Uczeń uzupełnia egzamin, utworzony przez nauczyciela |
| Warunki wstępne | Nauczyciel stworzył lekcję typu Egzamin Uczeń ma dostęp do kursu Egzamin jest aktywny |
| Główny przebieg zdarzeń | <ol style="list-style-type: none"> 1. Uczeń rozpoczyna egzamin 2. Uczeń rozwiązuje pytania egzaminacyjne 3. Uczeń zatwierdza ukończenie rozwiązywania egzaminu [A1: Termin wysłania minął] 4. Odpowiedzi zapisane są w systemie <p><u>Ocena egzaminu:</u></p> <ol style="list-style-type: none"> 5. System automatycznie ocenia zadania zamknięte na podstawie podanych przez nauczyciela odpowiedzi 6. Nauczyciel otrzymuje zadania otwarte do ocenienia 7. System sumuje punkty i wystawia ocenę |
| Alternatywny przebieg zdarzeń | <p>[A1: Termin wysłania minął]</p> <ol style="list-style-type: none"> 1. Uczeń otrzymuje informację o upływie czasu na wysłanie rozwiązania egzaminu 2. Odpowiedzi zapisane są w systemie 3. Nauczyciel otrzymuje informację o opóźnieniu wysłania zadania 4. Nauczyciel dopuszcza egzamin do oceny. [A2: Niedopuszczenie egzaminu] 5. Przejście do <u>Ocena egzaminu</u> <p>[A2: Niedopuszczenie egzaminu]</p> <ol style="list-style-type: none"> 1. Nauczyciel nie dopuszcza danego egzaminu do oceny 2. System wystawia uczniowi 0 punktów |

Tabela 4.5. Scenariusz przypadku użycia: Przeprowadzanie egzaminu

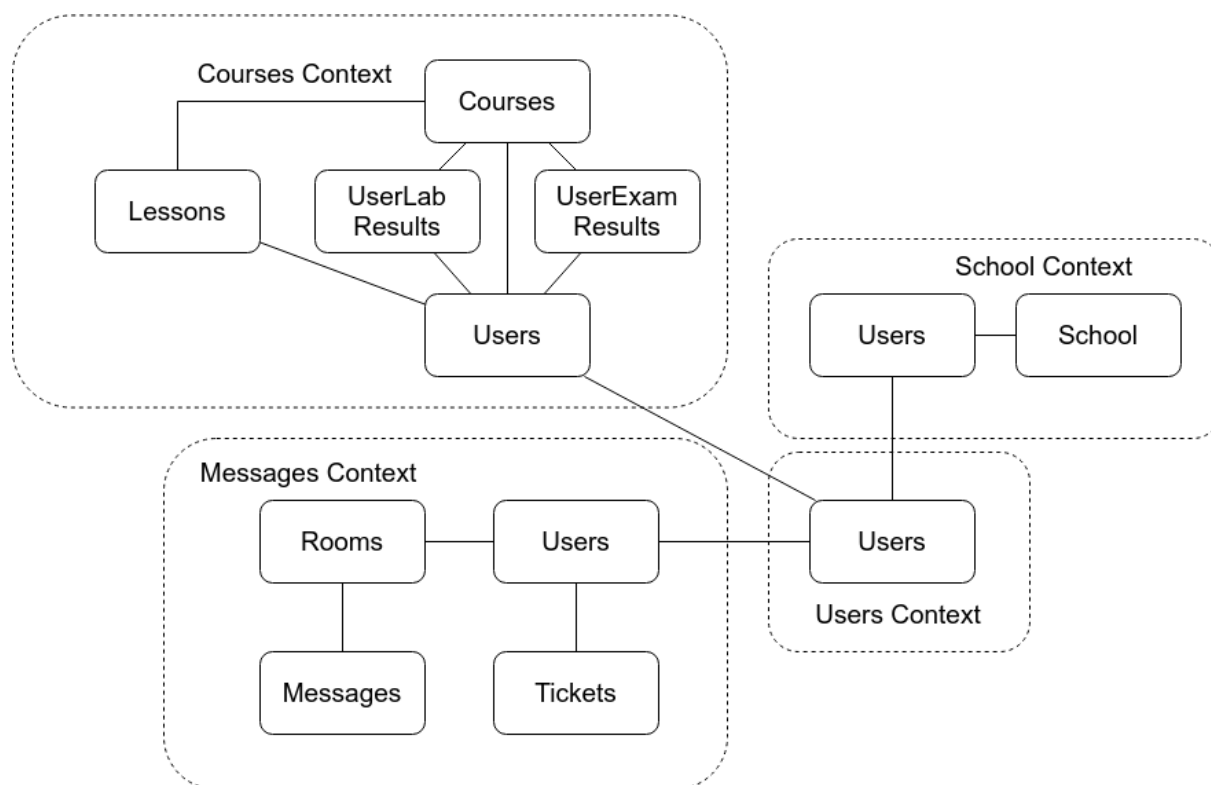
5. Projekt architektury

Niniejszy rozdział poświęcony jest omówieniu architektury projektowanego systemu. Przedstawiony zostanie sposób podziału systemu na mikroserwisy, omówiona będzie komunikacja między poszczególnymi elementami. W dalszej części rozdziału pokazane będą modele baz danych, wraz z opisem najistotniejszych pól. Ostatnim elementem rozdziału będzie przedstawienie sposobu autoryzacji użytkowników w systemie – omówiony zostanie framework OAuth2.0 wraz z OpenID Connect, a także tokeny JWT.

5.1. Architektura

5.1.1. Podział według kontekstów

Projektując system w oparciu o mikroserwisy, do najtrudniejszych zadań należy wydzielenie spójnych kontekstów między serwisami, tak aby serwisy mogły być autonomiczne i nie posiadały zbędnych danych. Na rysunku 5.1 przedstawiono podział danych na konteksty. Jak widać z diagramu, każdy kontekst posiada swoją własną encję użytkownika. Nie jest to jednak wszędzie ten sam użytkownik. W podejściu DDD, kontekst zawiera wszystkie potrzebne dane w ramach danej dziedziny i tylko te dane. Dlatego też użytkownik w kontekście kursów posiada inne właściwości niż użytkownicy w pozostałych dziedzinach. Oczywiście dojdzie do pewnych duplikacji danych, takich jak imię i nazwisko użytkownika, natomiast, jeżeli w ramach danego kontekstu są to dane uznane za wymagane, nie stanowi to problemu.



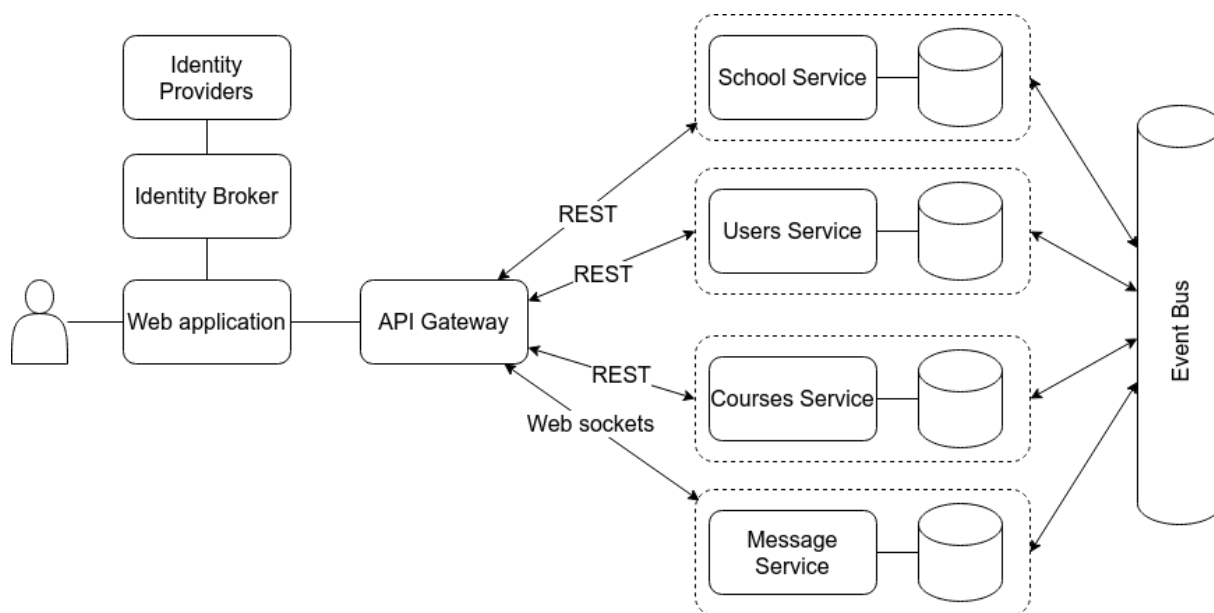
Rys. 5.1. Schemat podziału serwisów według kontekstów

Na powyższym diagramie wydzielono 4 konteksty. Pierwszym z nich jest kontekst użytkowników. Znajdują się tam wszelkie informacje o użytkowniku, zapis jego ustawień strony. Kontekst wiadomości definiuje potrzebne informacje dotyczące wysyłania wiadomości. Jest model wiadomości i pokoiów do których zapisani są użytkownicy. Kontekst kursów definiuje użytkownika jako uczestnika kursu. Na podstawie podziału kontekstów i granic między nimi zaprojektowano mikroserwisy obecne w tworzonej aplikacji.

5.1.2. Elementy systemu

Na rysunku 5.2 przedstawiono strukturę projektu. Wyszczególnione zostały 4 serwisy odpowiedzialne za logikę działania systemu. Komunikacja między nimi jest asynchroniczna, odbywa się przy użyciu magistrali zdarzeń (ang. event bus). Połączenie z aplikacją internetową możliwe jest dzięki zastosowaniu bramy interfejsu API. Ostatnim elementem systemu są aplikacje do uwierzytelniania użytkowników.

- Web application – Aplikacja uruchamiana w przeglądarce po stronie użytkownika. Odpowiedzialna jest za bezpośredni kontakt z użytkownikiem.
- Identity Broker – Ta część systemu odpowiada za uwierzytelnianie użytkowników. Umożliwia logowanie się w zarówno standardowy sposób – poprzez podanie loginu i hasła,



Rys. 5.2. Schemat systemu

jak i posiada możliwość połączenia się z aplikacjami zewnętrznymi, które uwierzytelniają użytkownika. Broker odpowiedzialny jest za wydanie tokenów, którymi użytkownik identyfikuje się, chcąc uzyskać dostęp do zasobów chronionych.

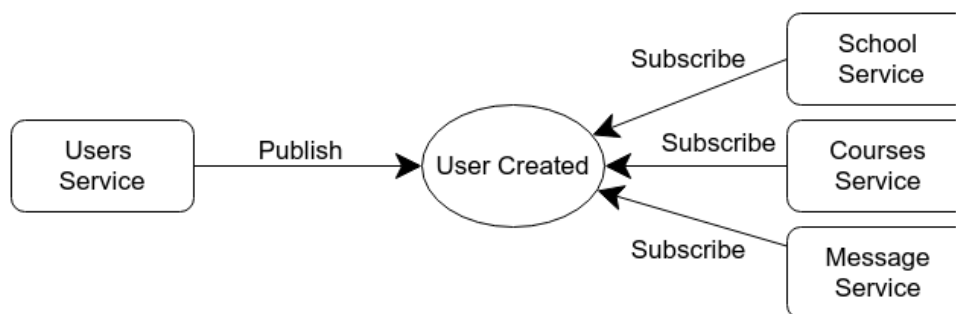
- Identity Providers – Aplikacje zewnętrzne implementujące standard OAuth2.0. Takimi aplikacjami są przykładowo Facebook, Twitter czy Github. Aplikacje te, po uzyskaniu odpowiedniej zgody, wydają token z informacjami o danych użytkownika i zakresie akcji, które aplikacja może wykonać w imieniu uwierzytelnionego użytkownika [14].
- API Gateway – Otrzymuje wszystkie żądania od aplikacji internetowej. Pełni rolę pośrednika między frontendem a backendem. Filtruje i przesyła dalej zapytania do poszczególnych serwisów. Dzięki zastosowaniu wzorca bramy mamy tylko jedno połączenie z aplikacją internetową. Upraszcza to implementację i utrzymanie systemu.
- School Service – Serwis do zarządzania właściwościami szkoły. Posiada moduł dla dyrektora, dzięki któremu może zarządzać rolami użytkowników w systemie.
- Users Service – Bazowy serwis. Posiada logikę do zarządzania użytkownikami. W momencie gdy nowy użytkownik zostaje utworzony, serwis wysyła wiadomość o danej akcji, dzięki czemu inne serwisy mogą dodać użytkownika do swoich baz danych.
- Courses Service – Serwis do obsługi kursów i tworzenia lekcji.

- Message Service – Serwis umożliwiający wysyłanie wiadomości przez użytkowników. Korzysta z technologii websocketów, dzięki czemu możliwa jest dwukierunkowa komunikacja w czasie rzeczywistym.
- Event Bus – Magistrala zdarzeń służy do komunikacji serwisów między sobą. Zapewnia asynchroniczne wysyłanie wiadomości. Serwis chcący coś wysłać, publikuje zdarzenie w magistrali, natomiast aby odebrać wiadomość, musi subskrybować dane zdarzenie.

5.1.3. Sposób komunikacji między serwisami

W architekturze mikroservisów wyróżnione są dwa sposoby komunikacji serwisów. Jest to *orkiestracja* i *choreografia*. Stosując orkiestrację mamy zazwyczaj serwis, posiadający informację o innych serwisach i ich funkcjonalnościach. Mając tą wiedzę wywołuje poszczególne serwisy do pracy. Powstaje w ten sposób wiele zależności między serwisami, co nie jest dobrym podejściem w architekturze mikroservisów.

Choreografia to podejście do łączenia mikroservisów w oparciu o zdarzenia. Serwis nie wywołuje poszczególnych serwisów do pracy, wysyła jedynie informację o zdarzeniu. Inne serwisy poprzez subskrypcję otrzymują wiadomość i mogą wykonać swoją logikę. Takie podejście zapewnia nam luźne powiązanie między serwisami. Serwisy mogą działać autonomicznie, są odporne na błędy innych serwisów w systemie. W projektowanym systemie wykorzystano podejście choreografii.



Rys. 5.3. Wykorzystanie choreografii podczas tworzenia użytkownika

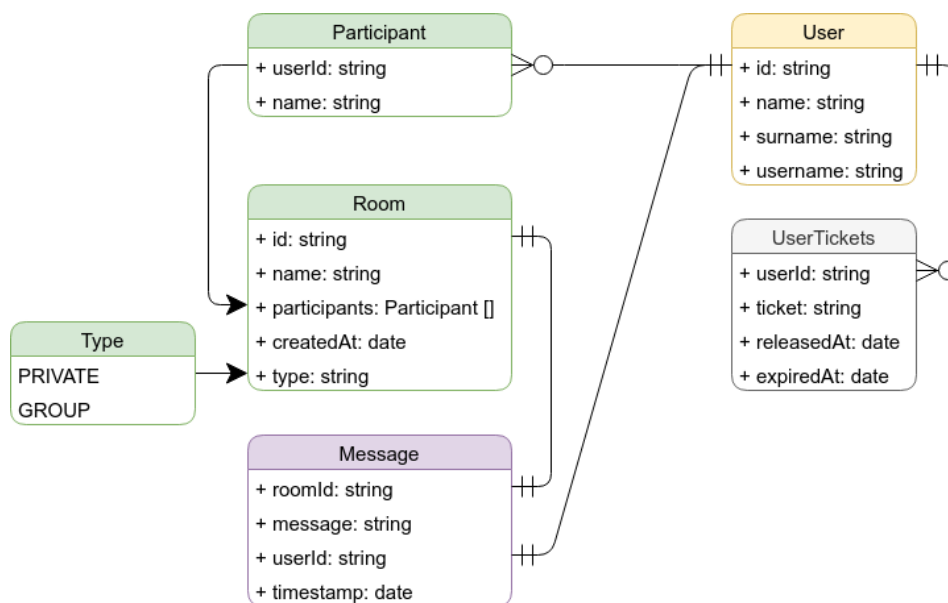
Na rysunku 5.3 przedstawiono sposób tworzenia użytkownika w systemie. Po rejestracji użytkownika tworzony jest odpowiedni rekord w serwisie szkoły. Następnie wysyłane jest zdarzenie informujące o tym fakcie. Serwisy które zasubskrybowały się do tego zdarzenia, po otrzymaniu informacji, tworzą użytkownika w swoich bazach. Serwis wysyłający powiadomienie nie wie, czy ktoś obserwuje dane zdarzenie, nie ma wiedzy o pozostałych serwisach. Takie podejście sprawia, że projektując nową mikrousługę nie musimy zmieniać logiki w pierwszym serwisie, wystarczy podłączyć się do nasłuchiwanego zdarzenia.

Na diagramie 5.4 przedstawiono model danych do zarządzania kursami. Wydzielone zostało 6 kolekcji:

- Users – Definiuje wszystkich użytkowników w systemie.
- Lessons – Kolekcja wszystkich lekcji w systemie. Ponieważ założenia systemu są takie, że lekcja nie jest własnością kursu, a osobnym niezależnym bytem, wydzielona została właśnie ta kolekcja. Lekcja to pojęcie abstrakcyjne, podzielona jest na 3 typy: egzamin, ćwiczenia i wykład. Dzięki zastosowaniu bazy nierelacyjnej, możliwe jest trzymanie wszystkich rodzajów lekcji w jednej kolekcji. Typ lekcji egzamin posiada w tym miejscu zbiór pytań różnego typu, wraz z odpowiedziami. Ćwiczenia mają dodatkowe pole opisujące zadanie. Wykład nie posiada specjalnych dla siebie pól.
- Courses – Kolekcja kursów. Oprócz podstawowych pól takich jak nazwa i opis posiada zbiór lekcji z wyżej omawianej kolekcji. Na tym etapie są to już instancje lekcji, należące do danego kursu. Nadpisowane są pola nazw i opisu, natomiast nauczyciel tworzący kurs, może je zmienić aby dopasować ją do kursu. Każdej lekcji w kursie można nadać daty dostępności a także komentować je. Egzaminom i ćwiczeniom ustawia się również skalę oceniania.
- UserCourses – Kolekcja w której trzymane są informacje o zarejestrowanych użytkownikach w ramach kursu. Ponieważ relacja użytkownik-kurs to relacja "many to many", wydzielona została właśnie ta kolekcja. Dzięki temu bardzo łatwo można uzyskać informacje jacy użytkownicy są zapisani w interesującym nas kursie, jak i na odwrót: w jakich kursach uczestniczy podany użytkownik.
- UserExamResults – w tej kolekcji przechowywane są wyniki egzaminów użytkowników. Dokument posiada odwołania do kolekcji użytkowników oraz kolekcji lekcji. Posiada również pole statusu, definiując stan dokumentu. Pierwszym stanem jest *GENERATED* - jest to początkowy stan, w którym pytania zostały przydzielone w ramach egzaminu i przesłane do studenta. Kolejnym stanem jest *RESOLVED*, pojawiający się w momencie przesłania odpowiedzi. Ostatnim stanem jest *ASSESSED*, czyli egzamin został oceniony.
- UserLabResults – Jest to podobna kolekcja do wcześniej wymienionej, przy czym zamiast egzaminów są ćwiczenia. Przechowywane są tam ścieżki do wysłanych plików wraz z informacją, kiedy zadanie zostało wysłane.

Dwoma dodatkowymi strukturami widocznymi z lewej strony diagramu są *Komentarze* i *Pliki*. Są to zagnieżdżane struktury w niektórych częściach modelu. Ze względu na częstotliwość ich pojawiania się w kolekcjach, brak jest strzałek, które mogłyby wpłynąć negatywnie na czytelność całości.

Kolejnym diagramem do omówienia jest model bazy danych dla serwisu do wysyłania wiadomości, przedstawiony na rysunku 5.5.

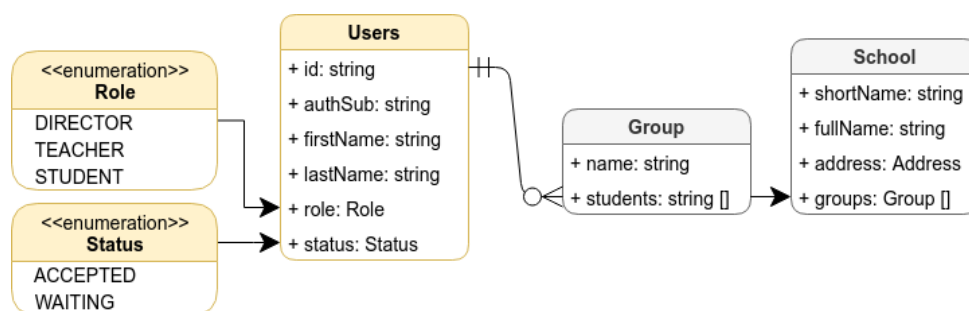


Rys. 5.5. Schemat bazy danych serwisu do wysyłania wiadomości

- Users – Przechowuje dane wszystkich użytkowników w systemie.
- UserTickets – Kolekcja przechowująca bilety wstępu do systemu. Serwis do wysyłania wiadomości działa w oparciu o technologie websocketów, stąd też potrzeba zdefiniowania specjalnego sposobu bezpiecznej komunikacji. Użytkownik chcąc połączyć się z serwisem wiadomości, przesyła swój token, a w zamian dostaje specjalny bilet, ważny przez określony czas. Następnie wysyła żądanie w celu dwukierunkowego połączenia przesyłając ten bilet. Po walidacji, system ustanawia połączenie.
- Rooms – Użytkownicy grupowani są w obrębie pokoi, gdzie mogą wysyłać sobie wiadomości. Pokój może być prywatny – czyli dla 2 osób, jak i grupowy - dla większej ilości osób. Typ ten został dodany, aby nie można było dodać trzeciej osoby do pokoju prywatnego. Podobnie działa Skype – w momencie gdy do prywatnej rozmowy zaprosimy trzecią osobę, nie zostanie ona dodana do tego pokoju, a zostanie utworzony nowy pokój, w którym nie będzie wcześniejszych prywatnych rozmów. Do pokoju typu Grupa, można bez żadnych przeszkód dodawać i usuwać użytkowników.

- Messages – Kolekcja wiadomości w systemie. Posiada referencję do pokoju w ramach którego została wysłana dana wiadomość, kto ją wysłał i kiedy. Alternatywą dla tej kolekcji było stworzenie zagnieżdżonej struktury w kolekcji pokoju, natomiast postanowiono stworzyć osobną kolekcję ze względu na ograniczenia pamięciowe bazy. W MongoDB pojedynczy dokument może mieć maksymalny rozmiar 16 megabajtów [15]. Biorąc pod uwagę, że wiadomości w ramach pokoju może być bardzo duża liczba, bezpieczniejszym rozwiązaniem jest wydzielenie osobnej kolekcji. W takim rozwiązaniu dokumentem jest pojedyncza wiadomość.

Ostatnim diagramem jest model bazy danych dla *School Service*. Posiada on 2 kolekcje: użytkowników oraz szkoły. Status użytkownika oznacza, czy dyrektor szkoły zatwierdził użytkownika w systemie, po zatwierdzeniu dostaje również odpowiednią rolę w systemie. Warto zauważyć, że użytkownik nie może mieć 2 ról jednocześnie, to znaczy być przykładowo studentem jak i nauczycielem.

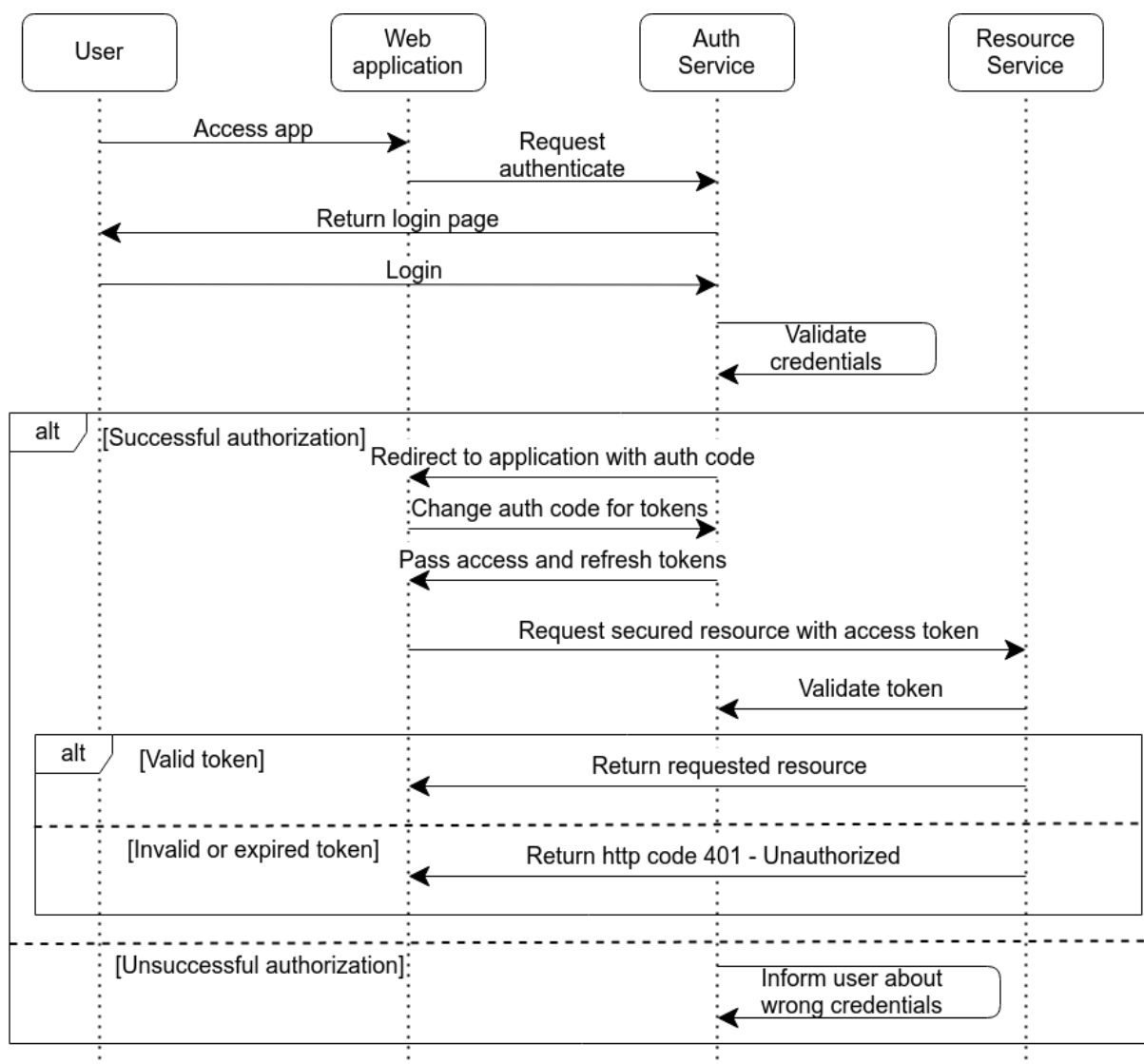


Rys. 5.6. Schemat bazy danych serwisu szkoły

5.3. Autoryzacja i uwierzytelnianie

Projektując aplikację internetową, do jednych z najważniejszych zadań należy zapewnienie bezpiecznego logowania do strony. Ponadto, w dzisiejszych czasach użytkownicy oczekują, że do strony będą mogli wygodnie zalogować się przy użyciu portali społecznościowych. Technologia która wychodzi naprzeciw tym oczekiwaniom jest OAuth2.0 – framework umożliwiający delegację autoryzacji do chronionych zasobów. OAuth 2.0 powstał w 2012 roku, zastępując wersję OAuth 1.0, z brakiem wstecznej kompatybilności [16].

Na diagramie 5.7 przedstawiono sposób działania OAuth 2.0.



Rys. 5.7. Sekwencja zdarzeń działania frameworka OAuth2.0

1. Użytkownik przechodzi na stronę internetową. W omawianym przypadku, ta strona jest klientem – czyli aplikacją chcącą uzyskać dostęp do zasobów chronionych.
2. Użytkownik wybiera opcję zalogowania się do strony. Zostaje przeniesiony do widoku serwisu autoryzacyjnego, gdzie może zalogować się poprzez podanie loginu i hasła, lub przy użyciu zewnętrznych aplikacji.
3. W przypadku wybrania drugiej opcji, użytkownik zostaje przeniesiony na stronę tej aplikacji, gdzie musi wyrazić zgodę na pobranie przez serwis autoryzacyjny potrzebnych danych.
4. Po udanym zalogowaniu, serwer autoryzacyjny przekierowuje użytkownika z powrotem do klienta, wraz z przesłaniem kodu autoryzacyjnego.
5. Klient po otrzymaniu kodu może wymienić go na tokeny. Przesyła go wraz z swoim identyfikatorem do serwisu autoryzacyjnego w celu uzyskania tokenów. Są to dwa tokeny: *Access token* oraz *Refresh token*. Access token wydawany jest na krótki okres czasu i służy do autoryzacji, natomiast refresh token wydawany jest na dłuższy okres, potrzebny jest do wygenerowania nowego access tokenu, gdy ten wygaśnie.
6. Serwis wydaje tokeny, dzięki którym klient może uzyskać dostęp do zasobów chronionych w serwerze.
7. Serwer z zasobami weryfikuje otrzymany token. Sprawdza czy jest poprawny oraz czy nie jest przedawniony. Jeżeli wszystko się zgadza, serwer zwraca odpowiedź z żądanymi danymi, w innym przypadku wysyła odpowiedni kod o braku autoryzacji.

Oauth 2.0 pozwala na autoryzację użytkowników. Nie jest on jednak przystosowany do uwierzytelniania. Protokołem, który to umożliwia jest OpenID Connect, będący dodatkową warstwą na Oauth 2.0. Standard ten definiuje dodatkowy token nazywany *ID Token* – posiada on dodatkowe informacje umożliwiające identyfikację użytkownika.

Tokeny zapisane są w formacie JWT (Json Web Token). Jest to otwarty standard umożliwiający przesyłanie danych między serwisami. Tokeny podpisywane są cyfrowo przy użyciu algorytmu HMAC lub RSA/ECDSA. Składa się z 3 części: nagłówka, głównej zawartości oraz sygnatury. Na nagłówek składa się informacja jakiego typu jest to token i w jaki sposób jest zakodowany. Następnie jest zawartość tokenu. Znajdują się tam między innymi dane użytkownika, jego role, czasy wydania i wygaśnięcia i inne. Ostatnią częścią tokenu JWT jest sygnatura, potwierdzająca autentyczność danych zawartych w tokenie [17].

eyJhbGciOiJSUzr1IiIsInR5cCI6I0AiSlldUWUi
ia2lkIiA6ICJYbDh6MmhhZ0RiZWktTGwYt1LlCRH
NoLTFwcFRYZXNyXzhxLTBnamM4TDNvIn0.eyJle
HAiOiJEMDg3MjY4NDcsIm1hdCI6MTYwODcxNTY0
NywiYXV0aF90aW11IjowLjQqdGkiOiI3YzE5OD
lOS0Z2ZjNjLTQwYmMtyjE5OS1mZlE2NzE4MjllMz
EiLCJpc3MiOiJodHRwOi8vbG9jYXRob3N0Ijgw
DvYXV0aC9yZWZsbnMvcmVtb3R1LXNjaG9vbC1y
ZWZsbnSiImF1ZCI6ImZyb250LWNsaWVudCisInN
1YiI6IjgyODNjNDk2LWY3Y2MtNGMwN05MzQ4LT
E0Yzc1MTJiOGVhNyIsInR5cCI6IklEiwiYXpwI
joiZnJvbnQtY2xpZW50Iiwic2VzC2lYb19zdGF0
ZSI6ImMyZjI4MzUxLWNhM2ItNDg4NS05OWR1LWI
0MGQwNTE1MGJlNyIsImF0X2hhc2giOiJwNEtGUm
9wazZDVEJ0Sk1jNTFjendBIiwiYWYyNyIjoiMSI
mVtYWlsX3Zlcm1maWVkiJjpmYWxzZSwicHJlZmV
cmVxX3VzZXJuYW11Ijoic3R1ZGVudCJ9.sa0taX
904ZUumCgPL0FgQrcI360IsjVIZdlNn4RnbMs3R
yWGAOcATKED0rRohr7dx7qrCXwiTej0RBZgetSs
qqLAWa40uVozGeCJSyeF9YvekX7SgEYbRluAXym
CJGKcIpi-
PSKdvEDA0UuUkYBd9C1Iv5K6Y4gCHV1-
Lm0ChQj1IHm8h6QCScGnB8iJtn0RUKcLB9MxsQpL
hhtBg6thi2oKCXPwrID9a9mW-
y8SLHf1kFL64TYW9bxbkPza7jUhk1q74jIIBtMGI
iCZiFUKAQ_4v0yGhUuMUZP7Nw3nj9ZByLgJE-
ajkYzZCcX-e9A2YFT32310PE37qyNMiB8vribuQ

```

HEADER: ALGORITHM & TOKEN TYPE

{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "Xl8z2hagDHyI-Ll2NYBDsh-1ppTXesr_8q-0gjc8L3o"
}

PAYLOAD: DATA

{
  "exp": 1608716847,
  "iat": 1608715647,
  "auth_ttime": 0,
  "jti": "7c1980e9-6f3c-40bc-b199-b31671829e31",
  "iss": "http://localhost:8080/auth/realms/remote-school-realm",
  "aud": "front-client",
  "sub": "8283c496-f7cc-4c04-9348-14c7512b8ea7",
  "typ": "ID",
  "azp": "front-client",
  "session_state": "c2f28351-ca3b-4885-99de-b40d05150be7",
  "at_hash": "p4KFRopk6CTBtJMc51czwA",
  "acr": "1",
  "email_verified": false,
  "preferred_username": "student"
}

VERIFY SIGNATURE

RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAAC
  AQ8AMIIBCgKCAQEAwMvBXTKbrDSj
  PhwW021i
  -----END PUBLIC KEY-----
)

```

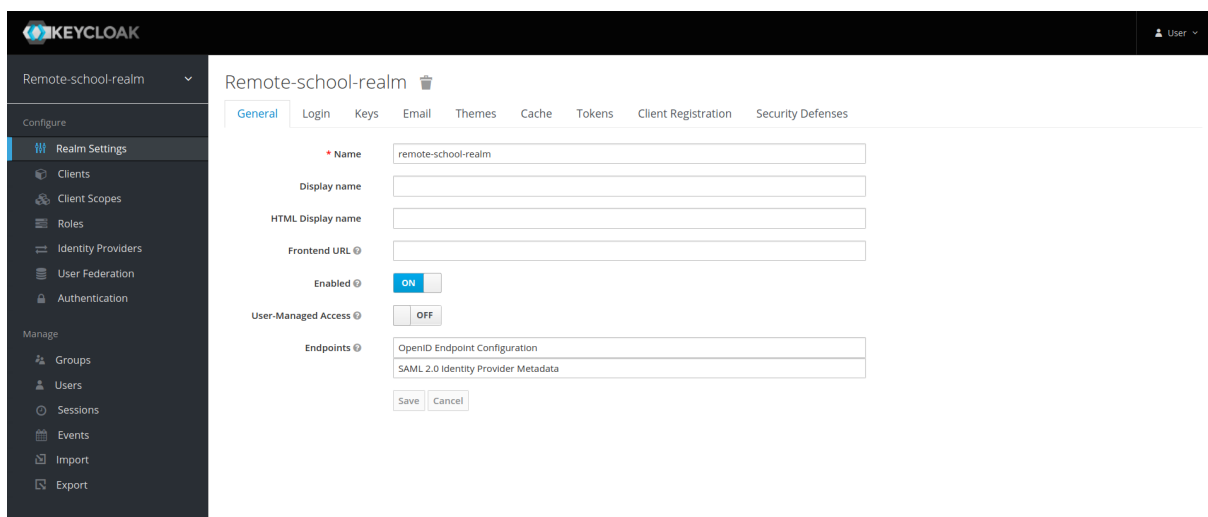
Rys. 5.8. Przykładowy token JWT

6. Implementacja

W tym rozdziale przedstawione zostaną szczegóły implementacyjne części systemu. Omówiony zostanie system wykorzystany do uwierzytelniania użytkowników. Następnie przedstawione będą szczegóły implementacyjne komunikacji asynchronicznej, integracji z bazą danych i bramy API, wraz z podaniem przykładowych listingów kodu. Ostatnią częścią rozdziału będzie przedstawienie struktury folderów mikroservisów.

6.1. Rejestracja, autoryzacja i uwierzytelnianie

We wcześniejszym rozdziale omówione zostały OAuth 2.0 oraz OpenId Connect, służące do autoryzacji i uwierzytelniania użytkowników. Narzędziem implementującym te rozwiązania jest Keycloak – otwartoźródłowe oprogramowanie tworzone przez firmę Red Hat. Do zadań Keycloaka należy między innymi rejestracja użytkowników, logowanie przy użyciu kont społecznościowych, logowanie SSO. Umożliwia również integrację z systemami LDAP.

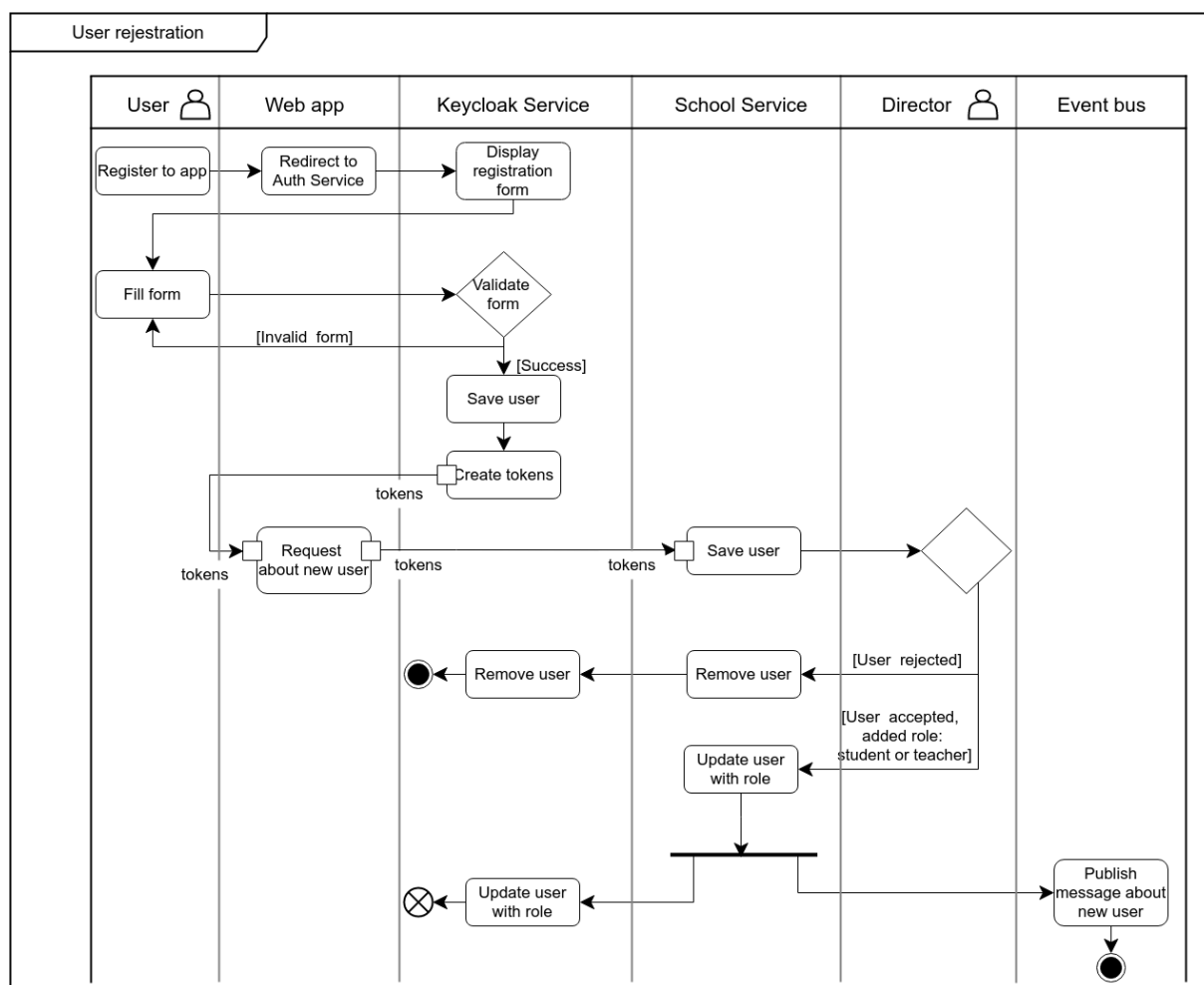


Rys. 6.1. Zrzut ekranu panelu administracyjnego Keycloak

W Keycloaku utworzone zostało królestwo (ang. realm) *Remote-school-realm*. Zawiera wszystkich użytkowników w systemie wraz z ich rolami. Zdefiniowani są również klienci, czyli

serwisy łączące się z keycloakiem. Każdy z mikroserwisów połączony jest jako *Bearer-only* – oznacza to, że serwisy nie odpowiadają za logowanie, a otrzymują one token i na jego podstawie decydują, czy użytkownik może wykonać żadaną akcję. Serwisy podczas uruchomienia, łączą się do keycloaka w celu uzyskania klucza. Dzięki takiemu rozwiązaniu, nie muszą potem łączyć się za każdym razem do keycloaka w celu weryfikacji tokenu, a robią to samodzielnie. Dodatkowo mikroserwis *School Service* posiada uprawnienia administratora keycloaka, dzięki czemu może dodawać role użytkowników. Dokładniej zostanie to omówione na rysunku 6.2, gdzie przedstawiony jest diagram czynności rejestracji nowych użytkowników.

Aplikacja internetowa połączona jest w sposób publiczny – użytkownik chcąc się zalogować z poziomu tej aplikacji, przenoszony jest do strony keycloaka, gdzie otrzymuje formularz logowania. Po udanym logowaniu, keycloak przekierowuje do zdefiniowanej wcześniej ścieżki w aplikacji wraz z kodem uwierzytelniającym [18].



Rys. 6.2. Rejestracja nowego użytkownika w systemie

Na diagramie 6.2 przedstawiona została sekwencja czynności rejestracji nowego użytkownika. Pierwszym etapem jest rejestracja użytkownika na stronie keycloak. Po udanej weryfikacji keycloak i wymianie z aplikacją internetową kodu uwierzytelniającego, przesyła tokeny.

Następnym krokiem jest przesłanie informacji do serwisu *School Service* o utworzeniu nowego użytkownika. Na tym etapie, użytkownik nie ma zdefiniowanej roli w systemie. Dyrektor otrzymuje listę nowo zarejestrowanych użytkowników może albo ich usunąć z serwisu, albo przydzielić odpowiednią rolę (student, uczeń, dyrektor). Po tej akcji, serwis przy pomocy API administratora wysyła odpowiednie dane do keycloak. Jeżeli dyrektor zatwierdził nowego użytkownika w systemie, wysyłane jest również zdarzenie asynchroniczne do innych serwisów, aby te mogłyby zaktualizować swoje bazy.

| Kod HTTP | Nazwa | Opis |
|----------|--------------|--|
| 201 | Created | Użytkownik dostał nową rolę w systemie |
| 400 | Bad Request | W przypadku podania niepoprawnej roli |
| 401 | Unauthorized | Niepoprawny lub przedawniony token |
| 403 | Forbidden | Brak odpowiedniej roli (dyrektor) |
| 404 | Not Found | Użytkownik nie został znaleziony |

Tabela 6.1. Możliwe odpowiedzi serwisu podczas dodawania roli

Autoryzacja i uwierzytelnianie użytkowników zostało zaimplementowane przy użyciu frameworka *Spring Security* z wykorzystaniem adaptera Keycloak dla języka Java[19]. Dla uwierzytelniania, serwisy wyciągają z tokenu przypisane role i walidują poprawność z tymi akceptowanymi przez serwis. W tym celu wykorzystano adnotacje *@RolesAllowed*.

```
@PostMapping
@RolesAllowed("teacher")
public void createLesson(@RequestBody CreatedLessonDto createdLessonDto) {
    logger.info("Create lesson");
    lessonsService.createCourse(authUser, createdLessonDto);
}
```

Listing 6.1. Wykorzystanie *@RolesAllowed* dla API tworzenia lekcji

6.2. API Gateway

API Gateway wykorzystane w obecnym projekcie zaimplementowane zostało z wykorzystaniem biblioteki Spring Cloud Gateway. Umożliwia ona na [20]. Przykładowe użycie bramy pokazano na listingu 6.2. Jest to fragment kodu z serwisu *API Gateway*, pokazujący jak są kierowane zapytania do odpowiednich serwisów. Każde zapytanie zaczynające się od */school* kierowane jest na port 8091, natomiast zapytania */users* na port 8092.

```
@Bean
public RouteLocator myRoutes(RouteLocatorBuilder builder) {
    return builder.routes()
        .route(p -> p
            .path("/school/**")
            .uri("http://localhost:8091"))
        .route(p -> p
            .path("/users/**")
            .uri("http://localhost:8092"))
}
```

Listing 6.2. Fragment API Gateway z Spring Cloud Gateway

6.3. Broker wiadomości

W rozdziale omawiającym projekt architektury omówiony sposób komunikacji serwisów między sobą. Odbywa się to asynchronicznie opartej na zdarzeniach. W projektowanym systemie użyto w tym celu RabbitMQ [21].

6.3.1. Wysłanie wiadomości

W momencie, gdy dyrektor zatwierdzi nowego użytkownika w systemie, wysyłane jest zdarzenie. W tym celu po stronie serwisu należy zdefiniować nazwę magistrali i klucza, dla którego wysłane zostanie zdarzenie.

```
@Configuration
public class RabbitConfiguration {
    @Bean
    public Exchange userCreatedEvent() {
        return new TopicExchange("userCreatedEvent");
    }
}

@Service
public class UserCreatedService {
```



```
// constructor and fields
public void createUserCreatedEvent(UserDto userDto) {
    String routingKey = "user.created";
    rabbitTemplate.convertAndSend(exchange.getName(), routingKey,
        userDto.toString());
}
}
```

Listing 6.3. Wysyłanie zdarzenia o nowym użytkowniku

6.3.2. Odebranie wiadomości

Serwis chcąc odebrać wiadomość musi zasubskrybować się do tej samej magistrali. Następnie tworzona jest kolejka wraz z powiązaniem jej do magistrali i klucza, dla którego odbierane są wiadomości. W omawianym przykładzie klucz ma nazwę *user.created* [22]. Definiowana jest również klasa, która odbiera wiadomość, następnie ją dekoduje i zapisuje w swojej bazie danych.

```
@Configuration
public class RabbitConfiguration {
    @Bean
    public Exchange eventExchange() {
        return new TopicExchange("userCreatedEvent");
    }
    @Bean
    public Queue queue() {return new Queue("QUEUE-NAME"); }
    @Bean
    public Binding binding(Queue queue, Exchange eventExchange) {
        return BindingBuilder.bind(queue).to(eventExchange)
            .with("user.created").noargs();
    }
    @Bean
    public UserCreatedConsumer eventReceiver() {
        return new UserCreatedConsumer();
    }
}

public class UserCreatedConsumer {
    @RabbitListener(queues = "QUEUE-NAME")
    public void createUser(String message) {
        //      decode and save user
    }
}
```

Listing 6.4. Odbieranie wiadomości z RabbitMQ

6.4. Komunikacja z bazą danych

Do integracji serwisów z bazą danych wykorzystano framework *Spring Data MongoDB*. Do podstawowych operacji typu CRUD, Spring generuje klasę *MongoRepository*. Tworząc takie repozytorium, wystarczy podać typ obiektu i klucza. Przykładowe repozytorium dla egzaminów pokazano na listingu 6.5.

```
public interface ExamRepo extends MongoRepository<Exam, String>{}
```

Listing 6.5. Przykład wykorzystania *MongoRepository* z *Spring Data Framework*

Jak widać podstawowe repozytorium, można stworzyć pisząc zaledwie jedną liniijkę. W momencie kiedy model jest bardziej zaawansowany i podstawowe operacje nie wystarczają, repozytorium rozszerzane jest o nową klasę, w której własnoręcznie należy zdefiniować metody.

```
public interface ExamRepo extends MongoRepository<Exam, String>,
    ExamCustomRepo {}

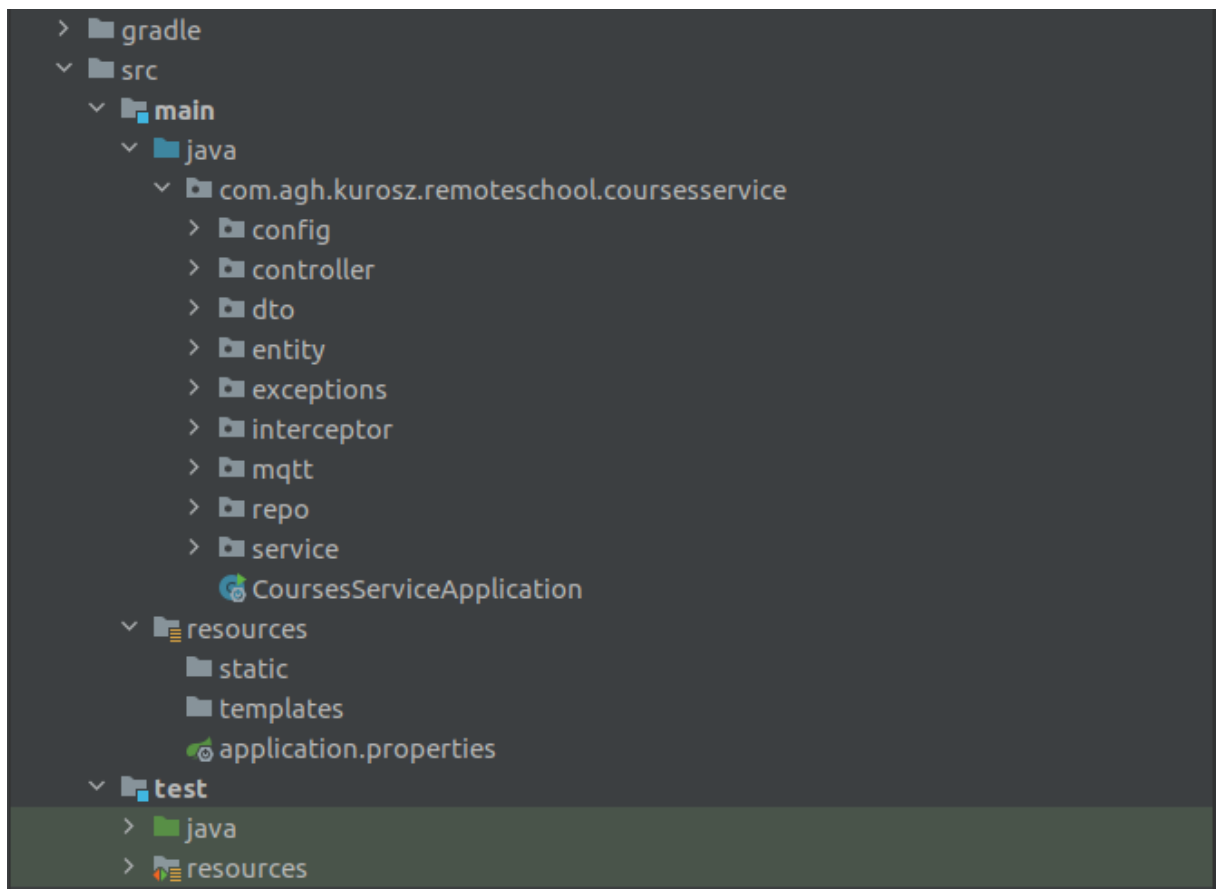
public interface ExamCustomRepo {
    void addQuestionToExam(String examId, List<Question> questions);
    Exam removeQuestionFromExam(String examId, String questionId);
    Question getQuestionById(String examId, String questionId);
}

@Repository
public class ExamCustomRepoImpl implements ExamCustomRepo {
    @Autowired
    protected MongoTemplate mongoTemplate;
    @Override
    public void addQuestionToExam(String examId, List<Question> questions){
        mongoTemplate.updateFirst(
            Query.query(Criteria.where("id").is(examId)),
            new Update().push("questions").each(questions),
            Exam.class);
    }
}
```

Listing 6.6. *ExamRepo* rozszerzone o dodatkowe metody

Repozytorium *ExamRepo* zostało rozszerzone o interfejs *ExamCustomRepo*, w którym zdefiniowano dodatkowe metody. Dodatkowo utworzona została klasa *ExamCustomRepoImpl*, implementująca poszczególne metody. Na listingu 6.6 pokazano przykład rozszerzenia interfejsu o 3 nowe metody, wraz z implementacją jednej z nich.

6.5. Struktura serwisów



Rys. 6.3. Struktura folderów w serwisie do zarządzania kursami

W folderze *config* znajdują się wszystkie pliki konfiguracyjne, między innymi bazy danych i keycloak. Kontrolery odpowiadają za wystawienie API restowego. Z kontrolerami łączy się kolejny folder: *dto*. DTO to klasy odpowiadające za wymianę informacji między serwisami [23]. Kontroler po otrzymaniu obiektu DTO, przesyła go dalej do Serwisu (folder *services*), gdzie mapowany jest na obiekt *entity*. Taki obiekt zostaje zapisany w bazie danych przy wykorzystaniu odpowiedniego Repozytorium. Pozostałymi folderami zdefiniowanymi w serwisie są

- *exceptions* – przechowywane są tam wyjątki, jakie aplikacja może wyrzucić,
- *mqtt* – połączenie z RabbitMQ,
- *interceptors* – zdefiniowane klasy wywoływane podczas żądania do kontrolera, na przykład *AuthInterceptor* odpowiedzialny za wstrzyknięcie danych uwierzytelniających użytkownika.

7. Wykorzystane technologie

7.1. Języki programowania

1. Java – obiektowy język programowania. Ze względu na prostotę i popularność języka, został wykorzystany do budowy mikroservisów w projektowanym systemie.
2. Typescript – Język będący nadzbiorem Javascript. Typescript umożliwia pisanie kodu w paradygmacie obiektowym, pozwala na typowanie. Typescript kompilowany jest do języka Javascript, dzięki czemu bez problemu uruchamiany jest w przeglądarkach.
3. HTML, CSS – podstawowe technologie wykorzystane do zbudowania aplikacji internetowej. W przypadku CSS wykorzystano preprocesor Sass.

7.2. Oprogramowanie, biblioteki i frameworki

1. Spring – framework języka Java. Wykorzystane elementy frameworka w projekcie to: Spring Boot, Spring Cloud, Spring Data oraz Spring Security.
2. Angular – utworzony przez Google framework, pozwalający budować aplikacje typu SPA (ang. Single Page Application). Angular został wykorzystany do zbudowania frontendu.
3. MongoDB – nierelacyjna baza danych oparta o dokumenty. Każdy mikroservis w zaprojektowanym systemie posiada swoją instancję tej bazy.
4. Keycloak – otwarto-źródłowe oprogramowanie IdAM (ang. Identity and Access Management), czyli narzędzie do zarządzania użytkownikami w systemie. Odpowiada za rejestrację, logowanie, połączenie z mediami społecznościowymi i wydawanie tokenów. Keycloak rozwijany jest przez JBoss firmy RedHat.
5. RabbitMQ – narzędzie umożliwiające kolejkowanie wiadomości. Wykorzystany w każdym mikroservisie do obsługi zdarzeń.

6. Stomp.js – biblioteka javascript umożliwiająca wykorzystanie technologii websockets. Wykorzystana po stronie frontendu do ustawienia połączenia.
7. ReactiveX – biblioteka wspierająca programowanie reaktywne, oparte o zdarzenia. Wykorzystana została wersja RxJs, czyli biblioteka przeznaczona dla języka Javascript.
8. Angular Material – biblioteka dostarczająca wiele komponentów interfejsu graficznego do użycia w projekcie pisanym w Angularze.

7.3. Narzędzia

1. IntelliJ IDEA Ultimate – środowisko programistyczne do pracy w języku Java od firmy JetBrains.
2. WebStorm – kolejne narzędzie firmy JetBrains. WebStorm pomaga w pracy nad frontendem aplikacji.
3. Robo 3T – aplikacja dostarcza interfejs graficzny do zarządzania bazą danych MongoDB.
4. Postman – narzędzie umożliwia testowanie API. Postman został wykorzystany do przygotowania kolekcji żądań, umożliwiających testowanie aplikacji.
5. draw.io – narzędzie online do tworzenia wykresów. W tym narzędziu przygotowane zostały wszystkie diagramy obecne w niniejszej pracy.

8. Interfejs systemu

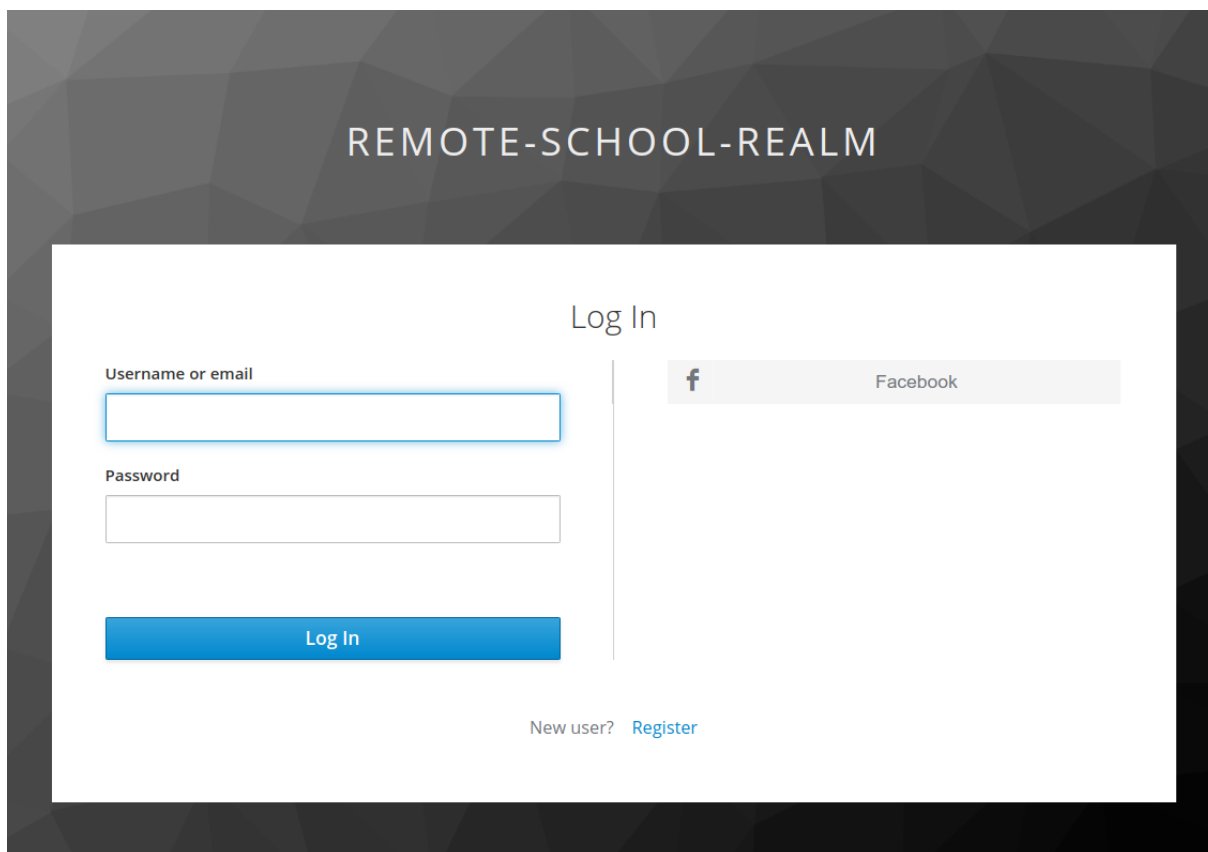
System do zdalnej edukacji powinien być przejrzysty i intuicyjny dla każdego użytkownika. Interfejs graficzny aplikacji powinna definiować prostota i minimalizm, tak aby nie rozpraszała użytkownika podczas nauki. Tymi zasadami kierowano się projektując poniższy interfejs [24].

Projektując aplikację wykorzystano ogólnodostępną bibliotekę komponentów dla aplikacji pisanych w Angularze, czyli Angular Material. Wykorzystanie biblioteki pomaga programiście w projektowaniu interfejsu, przyspieszając proces tworzenia komponentów. Aplikacje używające tej biblioteki charakteryzują się spójnym i estetycznym wyglądem, natomiast minusem jest to, że aplikacje korzystające z takich szablonów wyglądają do siebie podobnie, a sam proces dostosowania komponentu do swoich potrzeb bywa uciążliwy.

Kolorystyka interfejsu utrzymana została w trzech barwach: biały, niebieski oraz różowy. Inspiracja na takie barwy została zaczerpnięta z wymienionej wyżej biblioteki Angular Material, z której został wykorzystany motyw *Indigo & Pink*. Stonowane niebieskie i fioletowe barwy dobrze współgrają z jaskrawymi odcieniami czerwieni, pomarańczy i różu. Poprzez zastosowanie takiej palety barw, wzrok użytkownika od razu kieruje się na najważniejsze elementy strony, nie jest rozpraszany przez niepotrzebne dodatki.

Aplikacja została zaprojektowana jako aplikacja desktopowa. Style użyte w aplikacji pozwalają na pewną responsywność, natomiast chcąc używać aplikacji na telefonie, czy innym urządzeniu mobilnym, należałoby dokonać modyfikacji i poprawek. Aplikacja działa w każdej popularnej przeglądarce. Zrzuty ekranu zostały utworzone na podstawie wyświetlania systemu w przeglądarce Google Chrome.

Pierwszy zrzut ekranu (rys. 8.1) przedstawia stronę logowania do systemu. Jest to część systemu wygenerowana przez omówionego wyżej Keycloak, stąd też wygląda inaczej niż reszta aplikacji. Keycloak umożliwia zmianę motywu, natomiast w tym projekcie zdecydowano się na pozostawienie standardowej wersji. Lewa część strony to formularz logowania, gdzie użytkownik podaje login i hasło, obok jest miejsce na zintegrowane aplikacje zewnętrzne. W tym przypadku system połączono z Facebookiem. Poniżej znajduje się przycisk do rejestracji.



REMOTE-SCHOOL-REALM

Log In

Username or email

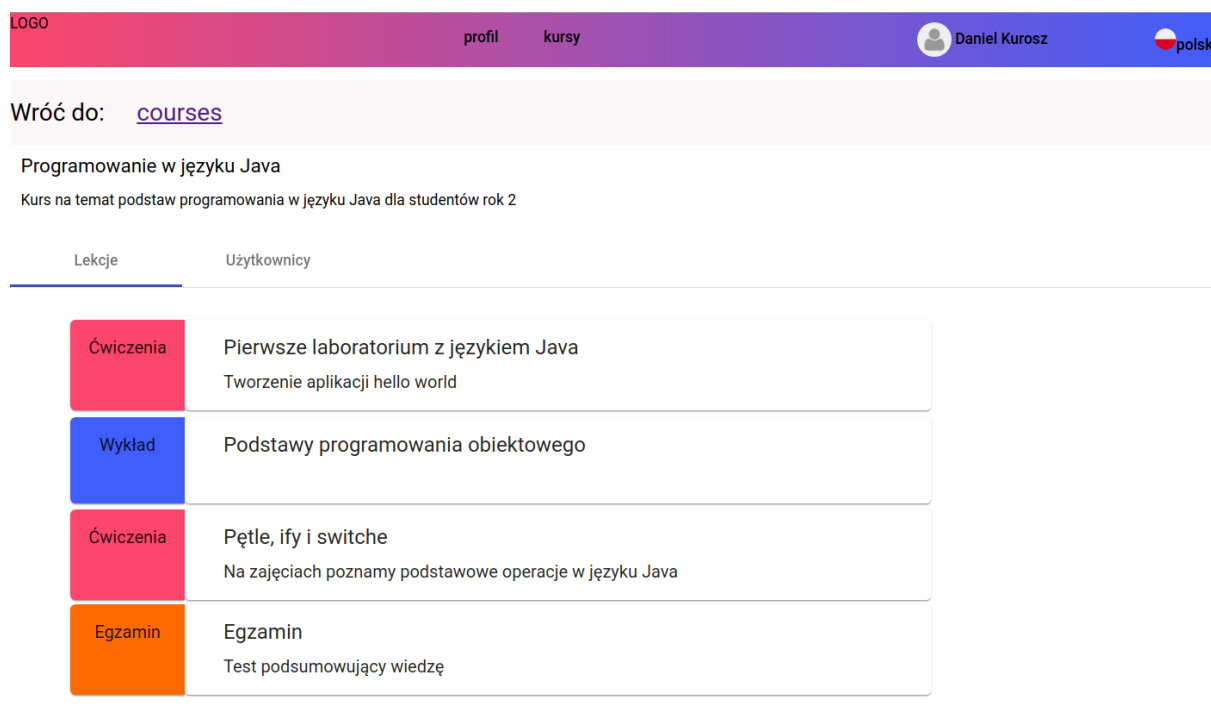
Password

Log In

New user? [Register](#)

Rys. 8.1. Interfejs aplikacji: logowanie do systemu

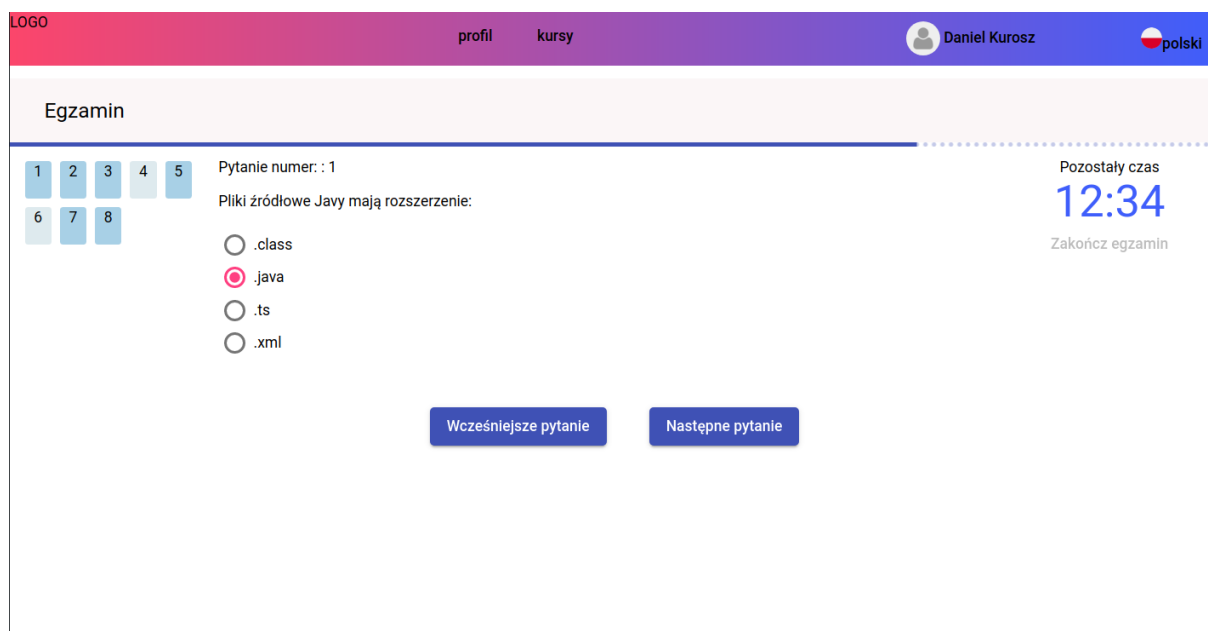
Rysunek 8.2 przedstawia aplikację podczas przeglądu lekcji wybranego kursu. Na samej górze znajduje się panel, z którego użytkownik może przejść do najważniejszych elementów systemu. W wersji prototypowej są 2 przyciski w menu: *profil* i *kursy*. Dalej znajduje się ikona z imieniem i nazwiskiem zalogowanego użytkownika oraz wybór języka strony. Kolejnym elementem jest tak zwany *okruszek* (ang. *breadcrumb*), czyli element na stronie, który zapamiętuje ścieżkę, jaką należy wykonać, aby przejść do danego widoku. Ułatwia to nawigację, aplikacja staje się wygodniejsza w użytkowaniu. Następnie znajdują się dane kursu: nazwa oraz krótki opis. Każdy kurs dzieli się na 2 zakładki: lekcje i użytkownicy. W zakładce użytkowników mamy listę zapisanych osób, w zakładce lekcji mamy listę lekcji (co widać na rysunku). Lekcje dzielą się na 3 typy, każdy z nich ma swój własny kolor, dzięki czemu użytkownik może szybciej zidentyfikować interesujący go element. W przyszłości należałoby również dodać informacje o datach dostępności danej lekcji, oraz zacieniować lekcje niedostępne w danym czasie.



Rys. 8.2. Interfejs aplikacji: przegląd lekcji w kursie

Kolejny rysunek przedstawia stan aplikacji podczas rozwiązywania egzaminu. Komponent składa się z 5 elementów:

- pasek stanu – oznacza procentowy stan rozwiązanych zadań. Początkowo w planie było stworzenie takiego paska stanu dla czasu, jaki pozostał uczniowi do zakończenia egzaminu, jednak widok takiego płynącego paska mógłby stresować ucznia, stąd też zdecydowano na utworzenie paska dla rozwiązanych zadań.
- lista pytań – są to małe prostokąty, z których użytkownik może wybierać do jakiego pytania chce przejść. Pytania już rozwiązane są pokolorowane ciemniejszym kolorem, dzięki czemu użytkownik wie, których zadań jeszcze nie rozwiązał.
- pytanie – główny element strony. Przedstawia treść pytania wraz z odpowiedziami.
- panel – na panelu umieszczono czas, informujący ile czasu zostało do zakończenia egzaminu, oraz przycisk do zatwierdzenia egzaminu. Ponieważ użytkownik nie rozwiązał wszystkich zadań, przycisk jest niedostępny.
- przyciski nawigacyjne – przyciski do nawigacji między pytaniami.



Rys. 8.3. Interfejs aplikacji: rozwiązywanie egzaminu

9. Testy

W niniejszym rozdziale omówiony zostanie sposób testowania implementowanego oprogramowania. Przedstawione zostaną testy jednostkowe, integracyjne oraz manualne.

Testy jednostkowe wykonywano w oparciu o technikę TDD (ang. Test Driven Development). Jest to sposób wytwarzania oprogramowania, w którym przed stworzeniem funkcjonalności przygotowujemy test. Metodykę TDD można opisać w 3 krokach:

1. Red – pierwsza faza cyklu. W tym kroku piszemy test, który zakończy się niepowodzeniem, z racji tego, że brak jest zaimplementowanej logiki. W momencie kiedy test na tej fazie zakończy się z sukcesem, jest to znak, że napisaliśmy błędnie sam test.
2. Green – w tym kroku napisana zostaje logika, aby test napisany w poprzedniej fazie zakończył się z powodzeniem. Na tym etapie kod powinien być napisany jak najprościej, kluczowe jest jedynie to, aby przeszedł test.
3. Refaktor – ostatni element cyklu TDD. Kod napisany w drugiej fazie powinien zostać zre-faktoryzowany, tak aby spełniał wszelkie zasady wytwarzania dobrego oprogramowania, był efektywny i optymalny.

Cykl ten należy powtarzać, aż wytworzy się pełną funkcjonalność serwisu [25].

```
@Test
public void addManyQuestionToRepo_ShouldReturnExamWithThreeQuestions() {

    addThreeQuestionsToExam();

    Exam exam = repo.findById(EXAM_ID).orElseThrow();

    assertEquals(3, exam.getQuestions().size());
}
```

Listing 9.1. Testy jednostkowe funkcjonowania bazy danych

Na listingu 9.1 przedstawiono przykładowy test jednostkowy sprawdzający funkcjonalność obsługi bazy danych. Testy pisane są w stylu *Given-when-then*. Oznacza on podział testu na 3 sekcje. W pierwszej sekcji definiowany jest stan systemu. Następnie w sekcji *When* wykonywany zostaje testowana funkcjonalność. W ostatniej fazie sprawdzana jest zgodność aplikacji według zdefiniowanych oczekiwań [26].

Oprócz testów jednostkowych wykorzystano testy integracyjne. Są to testy w których sprawdzany jest cały serwis. W tym celu należy mieć uruchomioną testową instancję serwisu na której uruchamiane są testy. Do najczęściej testowanej funkcjonalności było sprawdzenie, czy na po wysłaniu odpowiedniego żądania do bazy danych zostaną zapisane poprawne dane.

Podczas tworzenia oprogramowania korzystano również z testów manualnych. Po uruchomieniu serwisu, wykonywane zostało żądanie przy pomocy interfejsu Postmana i sprawdzana była odpowiedź. W taki sposób testowane było, czy działa poprawnie przesyłanie i walidowanie tokenów uwierzytelniających i czy dostęp do zasobów mają tylko użytkownicy z odpowiednimi rolami. Testy manualne wykonywane również były podczas testowania aplikacji pisanej w angularze. Testowane były w ten sposób komponenty, czy odpowiednio reagują na polecenia i czy nawigacja po systemie jest poprawna.

10. Wdrożenie aplikacji

Mikroserwisy zbudowane w ramach omawianego projektu działają w środowisku Spring. Są to aplikacje internetowe działające na kontenerze Apache Tomcat. Każdy serwis posiada swój własny kontener Tomcat. Każdy mikroserwis zbudowany został przy pomocy narzędzia Gradle. Jest to rozbudowane narzędzie do zarządzania zależnościami w projektach Javy, budowania projektu i zarządzania nim. Aplikacja internetowa pisana w Angular wymaga zainstalowanego Node.js. Podczas programowania wykorzystana została wersja node 10.20.0, a wersja Angular 9.1.12.

Każdy zbudowany serwis zostaje przekształcony na dockerowy obraz. Na listingu 10.1 przedstawiono zawartość pliku Dockerfile wraz z komendą, która buduje obraz dla danego mikroserwisu. Utworzony w ten sposób zostaje obraz zawierający system linux alpine wraz z zainstalowaną javą i zbudowanym serwisem. Dodatkowym atutem dockera jest fakt, że obrazy Keycloaka i bazy danych MongoDB można pobrać gotowe z repozytorium Docker Hub [27].

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]

docker build -t remote-school-courses .
```

Listing 10.1. Plik Dockerfile do budowania mikroserwisu wraz z komendą

Projektowany system jest w oparciu o mikroserwisy, co oznacza też, że działać będzie więcej niż jeden kontener. W tym celu przydatne jest narzędzie docker-compose, które umożliwia zdefiniowanie wielu obrazów dockera i połączeń. Na listingu 10.2 przedstawiony został fragment pliku docker-compose, w którym zdefiniowany został serwis do obsługi kursów wraz z bazą danych. W przypadku bazy danych definiowany jest obraz mongo (jest to obraz z Docker Hub), nazwa kontenera i port na którym działa baza. Następnie definiowany jest serwis.

Do zbudowania serwisu użyty jest obraz zbudowany na podstawie pliku i komendy z listingu 10.1, wskazane jest połączenie z bazą danych, oraz port na którym działa aplikacja. Plik docker-compose umożliwia wiele dodatkowych opcji, natomiast na etapie zaprojektowanego prototypu przedstawiony przykład jest wystarczający [28].

```
version: "3"
volumes:
  postgres_data:
    driver: local

services:

  mongodb-courses:
    image: mongo:4.2.8
    container_name: "mongo-courses"
    ports:
      - 27017:27017

  api-courses:
    image: remote-school-courses
    ports:
      - 9091:9090
    links:
      - mongodb-courses

//other services
```

Listing 10.2. Fragment pliku docker-compose do zbudowania systemu z wieloma kontenerami

11. Podsumowanie

11.1. Podsumowanie pracy

Celem niniejszej pracy był projekt systemu do zdalnej nauki wraz z implementacją prototypu. Projekt zakładał wykorzystanie nowoczesnych technik tworzenia oprogramowania takich jak mikroserwisy. Omawiana praca pozwoliła na rozwinięcie się w obszarach projektowania aplikacji, definiowania wymagań i funkcjonalności. Udało się stworzyć szkielety serwisów, pełną funkcjonalność logowania i rejestracji użytkowników wraz z zabezpieczeniem serwisów przy użyciu OAuth 2.0. Każdy mikroservis dostał solidną bazę, na której można implementować różne funkcjonalności. Zaprojektowane modele umożliwiają implementację wszystkich niezbędnych funkcjonalności, takich jak tworzenie lekcji, kursów, przeprowadzanie egzaminów i wysyłanie wiadomości.

11.2. Możliwy rozwój

Obecny stan techniczny nie pozwala na wdrożenie aplikacji jako gotowy produkt. Zostało dużo pracy dotyczącej implementacji funkcjonalności, połączenia aplikacji internetowej z mikroservisami. Zaprojektowany system jest bardzo rozległy, są to 4 mikroservisy i aplikacja frontendowa, implementacja całości wymagać będzie dużo czasu. Zaprojektowany system jest wystarczająco elastyczny; dzięki zastosowaniu mikroservisów, system można rozbudować o nowe serwisy. Nowymi mikroservisami, które można zaprojektować są przykładowo: serwis do wysyłania powiadomień czy serwis płatności, gdzie dyrektor mógłby zarządzać finansami szkoły i pensjami nauczycieli.

W dzisiejszych czasach coraz częściej użytkownicy konsumują treści poprzez urządzenia mobilne, więc należałoby również zaprojektować aplikację mobilną, oraz zmodyfikować obecny prototyp, tak aby był bardziej responsywny.

Spis rysunków

| | | |
|-----|--|----|
| 2.1 | Prognozowana wartość rynkowa branży LMS w latach 2018 - 2025 | 7 |
| 2.2 | Udział systemów LMS na rynku amerykańskim w latach 1997 - 2019 | 9 |
| 2.3 | Wykres migracji między systemami LMS w latach 2018 - 2020 | 10 |
| 3.1 | Porównanie systemu z niską kohezją i wieloma powiązaniem a systemem z wysoką kohezją i małą ilością zależności | 12 |
| 4.1 | Diagram przedstawiający logowanie do aplikacji | 17 |
| 4.2 | Diagram przypadków użycia dla zalogowanego użytkownika | 18 |
| 5.1 | Schemat podziału serwisów według kontekstów | 26 |
| 5.2 | Schemat systemu | 27 |
| 5.3 | Wykorzystanie choreografii podczas tworzenia użytkownika | 28 |
| 5.4 | Schemat bazy danych serwisu do zarządzania kursami | 29 |
| 5.5 | Schemat bazy danych serwisu do wysyłania wiadomości | 31 |
| 5.6 | Schemat bazy danych serwisu szkoły | 32 |
| 5.7 | Sekwencja zdarzeń działania frameworka OAuth2.0 | 33 |
| 5.8 | Przykładowy token JWT | 35 |
| 6.1 | Zrzut ekranu panelu administracyjnego Keycloak | 37 |
| 6.2 | Rejestracja nowego użytkownika w systemie | 38 |
| 6.3 | Struktura folderów w serwisie do zarządzania kursami | 43 |
| 8.1 | Interfejs aplikacji: logowanie do systemu | 48 |
| 8.2 | Interfejs aplikacji: przegląd lekcji w kursie | 49 |
| 8.3 | Interfejs aplikacji: rozwiązywanie egzaminu | 50 |

Spis tablic

| | | |
|-----|--|----|
| 4.1 | Scenariusz przypadku użycia: Rejestracja nowych użytkowników | 19 |
| 4.2 | Scenariusz przypadku użycia: Tworzenie nowej lekcji w systemie | 20 |
| 4.3 | Scenariusz przypadku użycia: Tworzenie kursu w systemie | 21 |
| 4.4 | Scenariusz przypadku użycia: Przesyłanie zadań | 22 |
| 4.5 | Scenariusz przypadku użycia: Przeprowadzanie egzaminu | 23 |
| 6.1 | Możliwe odpowiedzi serwisu podczas dodawania roli | 39 |

Bibliografia

- [1] *What is an LMS?* URL: <https://www.valamis.com/hub/what-is-an-lms#what-is-lms> (term. wiz. 2020-10-16).
- [2] Markets i Markets. *LMS Market by Component (Solution and Services), Delivery Mode (Distance Learning, Instructor-Led Training and Blended Learning), Deployment Type, User Type (Academic and Corporate), and Region - Global Forecast to 2025*. 2020. URL: <https://www.marketsandmarkets.com/Market-Reports/learning-management-systems-market-1266.html> (term. wiz. 2020-11-12).
- [3] URL: <https://upel.agh.edu.pl/kokpit/obszary.html> (term. wiz. 2020-10-16).
- [4] URL: <https://stats.moodle.org/> (term. wiz. 2020-10-16).
- [5] URL: <https://www.getapp.com/education-childcare-software/a/moodle/reviews/> (term. wiz. 2020-10-16).
- [6] URL: <https://www.blackboard.com/pl-pl> (term. wiz. 2020-10-16).
- [7] URL: <https://philonedtech.com/lms-market-trends-a-view-of-recent-migrations-outside-north-america/> (term. wiz. 2020-10-16).
- [8] URL: <https://github.com/instructure/canvas-lms> (term. wiz. 2020-10-16).
- [9] S. Newman. *Building Microservices*. O'Reilly Media, 2015.
- [10] K. Sacha. *Inżynieria oprogramowania*. Wydawnictwo naukowe PWN, 2010.
- [11] URL: <https://www.trustradius.com/products/blackboard-learn/reviews?qs=pros-and-cons> (term. wiz. 2020-10-16).
- [12] R. Martin. *Czysty kod - Podręcznik dobrego programisty*. Helion, 2010.
- [13] Marcinkowski B. Wyrzykowski K. Wrycza S. *Język UML 2.0 w modelowaniu systemów informatycznych*. Helion, 2006.
- [14] URL: <https://oauth.net/2/scope/> (term. wiz. 2020-12-16).
- [15] URL: <https://docs.mongodb.com/manual/reference/limits/> (term. wiz. 2020-12-16).

- [16] URL: <https://oauth.net/2/> (term. wiz. 2020-12-16).
- [17] URL: <https://jwt.io/introduction> (term. wiz. 2020-12-16).
- [18] URL: https://www.keycloak.org/docs/latest/securing_apps/ (term. wiz. 2020-12-16).
- [19] URL: https://www.keycloak.org/docs/latest/securing_apps/#_spring_boot_adapter (term. wiz. 2020-12-16).
- [20] URL: <https://spring.io/projects/spring-cloud-gateway> (term. wiz. 2020-12-16).
- [21] URL: <https://www.rabbitmq.com/> (term. wiz. 2020-12-12).
- [22] URL: <https://www.rabbitmq.com/tutorials/tutorial-five-python.html> (term. wiz. 2020-12-16).
- [23] URL: <https://martinfowler.com/eaCatalog/dataTransferObject.html> (term. wiz. 2020-12-16).
- [24] URL: <https://www.nngroup.com/articles/ten-usability-heuristics/> (term. wiz. 2020-12-16).
- [25] K. Beck. *TDD. Sztuka tworzenia dobrego kodu*. Helion, 2014.
- [26] URL: <https://martinfowler.com/bliki/GivenWhenThen.html> (term. wiz. 2021-01-03).
- [27] URL: <https://hub.docker.com/> (term. wiz. 2021-01-03).
- [28] URL: <https://developer.okta.com/blog/2019/02/28/spring-microservices-docker> (term. wiz. 2021-01-03).