POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Krzysztof Gawliński
Nr albumu 16130

Rozszerzalna platforma zapobiegająca oszustwom podczas egzaminowania online

Praca magisterska napisana
pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, wrzesień 2018

Streszczenie

Praca dotyczy problemu oszukiwania podczas rozwiązywania egzaminów na platformach internetowych. Istniejące rozwiązania umożliwiają nadzór nad osobą egzaminowaną. W tym dokumencie została przedstawiona koncepcja dotycząca innego spojrzenia na to zagadnienie.

W ramach pracy powstał działający prototyp z nazwą roboczą Legito, który umożliwia instytucjom potrzebującym tego typu rozwiązań, przeprowadzenie egzaminów w sposób bezpieczny, eliminujący podstawowe oszustwa.

Praca rozpoczyna się od przedstawienia jej celu, oczekiwanych wyników i opisu poszczególnych rozdziałów. Zostały w niej omówione istniejące rozwiązania, podobne do prototypu lub rozwiązujące ten sam problem - oszustwa podczas wypełniania egzaminów online. Opisane są dwie koncepcje podejścia do tego tematu i wnioski, na bazie których zapadła decyzja o wyborze jednej z nich. Przedstawiony jest użyty stos technologiczny wraz z opisem poszczególnych narzędzi. Architektura systemu została opisana na podstawie objaśnienia roli w systemie, poszczególnych jego elementów. W podsumowaniu rozdziału dotyczącego architektury zostały zaprezentowane wnioski o użyciu mikroserwisów do stworzenia prototypu. W pracy znajduje się szczegółowy opis funkcjonalności oferowanych przez system. Autor przybliża również plany rozwojowe m.in. re factoring oraz nowe funkcjonalności.

Spis treści

1. Wstęp

Konieczność przeprowadzania testów internetowych jest nie tylko sposobem na oszczędność papieru, i tym samym naszego środowiska. Dzięki tego typu rozwiązaniom, egzaminatorzy poświęcają dużo mniej czasu na przeglądanie każdej pracy, zwłaszcza jeśli jest to na przykład test wyłącznie wielokrotnego wyboru. Stworzone w niniejszej pracy rozwiązanie ma na celu przeciwdziałać oszustwom, podczas zdalnego wypełniania egzaminów.

1.1. Cel pracy

Celem pracy jest przybliżenie i próba rozwiązania problemu oszustw podczas egzaminów wypełnianych online. W ramach pracy, na podstawie opisanej w niej koncepcji, ma powstać prototyp umożliwiający bezpieczne zdalne przeprowadzanie testów.

1.2. Rozwiązania w pracy

Platforma jest zaprojektowana w taki sposób, aby móc obsługiwać duże obciążenia i przetwarzać znaczące ilości danych. W tym celu została wykorzystana architektura mikroserwisów (z ang. microservices architecture).

Oprogramowanie zostało napisane w języku C# i wykorzystuje najnowszy dostępny w obecnym czasie Framework .NET Core 2.0. Dla optymalizacji wykorzystano zarówno bazy relacyjne SQL Server, jak i bazę NoSQL - RavenDB.

Dla skalowalności rozwiązania wszystkie usługi zostały zamknięte w kontenerach Docker, bazujących na systemie operacyjnym Linux. Podczas startu rejestrują się one w programie umożliwiającym odkrywanie usług - aplikacji Consul (HashiCorp). Komunikacja między nimi odbywa się poprzez protokół HTTP i kolejkę RabbitMQ.

W celu zabezpieczenia API, znajdujących się w usługach, została wykorzystana biblioteka IdentityServer4, która umożliwia autentykacje użytkowników i autoryzację pomiędzy poszczególnymi aplikacjami.

Dla optymalnego wykorzystania API, warstwa interfejsu użytkownika została napisana w języku TypeScript z użyciem biblioteki Angular.

1.3. Rezultat pracy

Rezultatem pracy jest koncepcja systemu o roboczej nazwie Legito, który umożliwia tworzenie, wypełnianie egzaminów oraz procesowanie ich pod kątem oszustw. Dzięki wykorzystanej architekturze platforma może być rozszerzana przez programistów o kolejne elementy skanujące prace studentów.

System składa się z następujących elementów:

- Usługi zarządzającej wzorami i rozwiązaniami egzaminów,
- Usługi zarządzającej autoryzacją, autentykacją i użytkownikami,

- Usługi zarządzającej komunikacją między warstwą interfejsu użytkownika, a innymi elementami systemu,
- Usługi koordynującej pracę procesorów,
- Procesora sprawdzającego podobieństwo odpowiedzi na konkretne pytanie pomiędzy studentami wypełniającymi ten sam egzamin,
- Procesora sprawdzającego czy odpowiedź nie została skopiowana z Wikipedii,
- Aplikacji interfejsu użytkownika,
- Kolejki RabbitMQ,
- Odkrywanie usług Consul,
- Elasticsearch,
- Kibana.

1.4. Organizacja pracy

Praca rozpoczyna się od krótkiego omówienia oprogramowania wykorzystywanego w celu zapobiegania oszustwom, dostępnego obecnie na rynku. Autor przedstawia w niej wybrane rozwiązania, które są zbliżone funkcjonalnie, oferują pojedynczą funkcjonalność zawartą w prototypie lub rozwiązują ten sam problem.

Rozdział trzeci przedstawia propozycję budowy systemu, założenia, które powinna spełniać aplikacja oraz wstępny jej zarys. Poruszony zostanie temat zalet i wad egzaminowania online.

Rozdział czwarty opisuje wszystkie narzędzia i technologie jakie zostały wykorzystane do budowy prototypu.

W rozdziale piątym omówione zostaną techniczne rozwiązania przyjęte w pracy. Zostaną przedstawione wzorce projektowe i przepływy, przybliżone pojęcia odkrywania usług i mikroserwisów. Zademonstrowana będzie możliwość rozszerzenia platformy o kolejne procesory zapobiegające oszustwom, a także hostowanie aplikacji na środowisku deweloperskim.

Rozdział szósty szczegółowo opisuje wszystkie funkcjonalności zawarte w systemie oraz sposób ich wykorzystania.

Rozdział siódmy to plany rozwojowe platformy. Poruszone będą tematy takie jak implementacja procesów CI/CD (z ang. continuous integration / continuous delivery), czy pomysł na Framework dla procesorów. Zostaną wskazane miejsca, które mogą zostać poddane re factoringowi.

Rozdział ósmy, jako ostatni, dotyczy podsumowania prac i będzie prezentować wnioski płynące z pracy nad systemem.

2. Istniejące rozwiązania

Autor nie odnalazł systemu, który odpowiadałby w pełni stworzonemu na potrzeby niniejszej pracy prototypowi. Uczelnie, i nie tylko one, wykorzystują oprogramowanie, które pomaga jedynie w nadzorze podczas prowadzenia egzaminów, czym starają się rozwiązywać problem oszustwa. Opis najpopularniejszych systemów zostanie przedstawiony w kolejnych podrozdziałach. Oprócz tego, opisane zostaną ich funkcjonalności i różnice, występujące w stosunku do prototypu.

2.1. Proctoru

Najczęściej najprostsze rozwiązania są wystarczająco dobre. W taki sposób podchodzi do tego firma zajmująca się oprogramowaniem Proctoru [36], która na swojej platformie wychodzi z założenia, że najlepszym sposobem na powstrzymanie ludzi przed oszukiwaniem na egzaminach jest ich nadzór przez Internet.

Z perspektywy studenta proces wygląda następująco:

- Rezerwacja terminu,
- W czasie, w którym wypada rezerwacja, wybranie opcji wypełniania egzaminu,
- Uzyskanie połączenia z osobą nadzorującą,
- Weryfikacja danych osobowych,
- Udostępnienie pulpitu,
- Weryfikacja użytkownika,
- Postępowanie zgodnie z instrukcjami osoby nadzorującej.

Działanie aplikacji jest proste. Osoba musi posiadać konto i być zalogowanym. W umówionym czasie można podejść do egzaminu, podczas którego następuje połączenie z osobą z firmy Proctoru, poprzez kamerę internetową. Udostępniany jest również pulpit studenta. Nadzorujący nakazuje pokazanie otoczenia osobny egzaminowanej i weryfikuje jej tożsamość. Po uzyskaniu jego zgody test może zostać rozpoczęty. Nadzorujący jest dostępny podczas całego procesu egzaminowania.

Podczas wypełniania egzaminu osoba zdająca powinna znajdować się w cichym i dobrze oświetlonym pokoju. Musi też mieć pod ręką dokument potwierdzający tożsamość. Niedozwolone jest używanie jakichkolwiek zewnętrznych programów. Komputer nie może posiadać więcej niż jeden monitor. Zabronione jest korzystanie z telefonów komórkowych i smartfonów. Nie wolno również zakrywać twarzy.

System nie jest w żadnym stopniu podobny do prototypu - z jednym wyjątkiem - stara się rozwiązywać ten sam problem. Jest on zintegrowany z wieloma systemami zarządzania nauczaniem poprzez wykorzystanie protokołu LTI (*z ang. Learning Tools Interoperability*). Sama aplikacja (czy model biznesowy) skaluje się w bardzo słaby sposób. Ilość osób, które mogą być egzaminowane w jednym momencie, jest ograniczona nie poprzez wydajność systemu, lecz ilość osób zatrudnionych do nadzoru. Jest to potencjalnie największa jego wada.

Rozwiązanie przyjęte w systemie jest proste. Pomimo tego aplikacja jest chętnie wykorzystywana przez placówki akademickie i inne organizacje. Najważniejsze benefity jakie niesie ze sobą egzaminowanie przez Internet, jakie są w niej obecne:

- Brak konieczności dojazdu,
- Rozwiązanie wymagające jedynie dostępu komputera,
- Możliwość wybrania dogodnego terminu egzaminu.

Jej unikalną funkcjonalnością jest właśnie możliwość wybrania terminu, który pasuje studentowi. W żadnym innym systemie znalezionym przez autora, ani nawet w prototypie, nie jest to możliwe.

W przeciwieństwie do prototypu, rozwiązanie jest produktem komercyjnym i nie może być w łatwy sposób rozszerzane przez zewnętrzne osoby. Nadzór nad kodem sprawuje producent. Do tworzenia egzaminów trzeba wykorzystywać zewnętrzny system.

2.2. AIProctor

System reklamowany jako najbardziej zaawansowany pod względem technologicznym istniejący na rynku, i w najbardziej przystępnej cenie. W swoim działaniu wykorzystuje algorytmy sztucznej inteligencji i zdalny nadzór.

AIProctor [37] jest aplikacją bardzo prostą w użytkowaniu, wygląda jakby ta prostota właśnie była głównym celem biznesowym producenta. Po udanej integracji z systemem zarządzania nauczaniem, proces egzaminowania (od początku do końca) wygląda następująco:

- Zapłata za egzaminowanie,
- Tworzenie egzaminu,
- Włączenie wybranych funkcjonalności uniemożliwiających oszustwa,
- Zainstalowanie rozszerzenia przeglądarki,
- Weryfikacja studenta,
- Odebranie wyników.

Przed decyzją o korzystaniu z systemu przysługuje okres próbny. Po jego zakończeniu należy opłacać jego wykorzystanie zgodnie z cennikiem przewidzianym przez producenta. Egzaminator loguje się do systemu zarządzania nauczaniem i w nim tworzy egzamin. Włączanie funkcji zabezpieczających odbywa się za pomocą jednego kliknięcia w każdą z nich, co jest bardzo prostym rozwiązaniem. Wtyczka instalowana jest z poziomu przeglądarki Chrome i jest ona wymagana do poprawnego wypełnienia egzaminu. Weryfikacja egzaminowanego jest przeprowadzana na podstawie skanowania twarzy, przedstawienia dokumentu i skanowania pokoju, w którym student się znajduje, w perspektywie 360 stopni. Wyniki egzaminu są wysyłane bezpośrednio do systemu zarządzania nauczaniem, dzięki czemu nie ma potrzeby logowania do innej platformy.

Niektóre z zalet systemu:

- Łatwa instalacja,

- Inteligentne blokowanie przeglądarki,
- Inteligentne oznaczanie egzaminów,
- Nie wymaga logowania.

Instalacja przebiega na podstawie instalacji wtyczki do przeglądarki Chrome, nie ma potrzeby instalować żadnego innego oprogramowania i jego mozolnej konfiguracji. Blokowanie przeglądarki polega na wyłączaniu poszczególnych funkcji, dzięki czemu osoba egzaminowana ma dostęp do niewielkiej ilości możliwych akcji podczas wypełniania testu. Oznaczanie egzaminów opiera się na zapisywaniu podejrzanych działań. Dzięki temu nadzorujący są w stanie w łatwy sposób zweryfikować oszustwa podczas całego procesu. AIProctor integruje się z systemami zarządzania nauczaniem. Wystarczy do tego kilka kliknięć.

Oprócz powyższych, system ten posiada inne zaawansowane funkcje. Przykładem może być nadzór, dostępny z kilku urządzeń jednocześnie. Mogą to być smartfony lub tablety, rozmieszczone w pokoju pod różnymi kątami. Oprócz monitorowania działań na komputerze lub telefonie, umożliwia detekcję podejrzanych działań z wykorzystaniem smartfonów. Pozwala na śledzenie osoby wypełniającej egzamin, a dokładniej określenie jej precyzyjnej lokalizacji. Dzięki temu, instytucja zlecająca egzamin może kontrolować obszar, z którego jest on wypełniany. Możliwe jest też wykrywanie dodatkowych urządzeń, co pozwala blokować kolejne możliwości oszustwa.

Aplikacja jest z pewnością bardziej zaawansowana niż stworzony prototyp. Tak jak inne oprogramowania do nadzoru, skupia się na jak najdokładniejszym nagrywaniu osoby wypełniającej egzamin, i uniemożliwia korzystanie z urządzeń innych niż komputer służący do wypełniania egzaminu. Blokuje również akcje, których nie życzy sobie instytucja zlecająca egzamin.

W przeciwieństwie do prototypu, nie posiada funkcjonalności skanujących egzaminy już wypełnione. Wyłapuje i oznacza niepożądane zachowania, które muszą następnie zostać ocenione przez fizycznych pracowników firmy. Do tworzenia egzaminów konieczne jest skorzystanie z zewnętrznych narzędzi. Jest to platforma komercyjna, więc jej rozszerzalność pod względem funkcjonalności jest zależna tylko i wyłącznie od producenta oprogramowania.

2.3. Proctortrack

Według producenta jest to system najbardziej przyjazny studentowi. Jest wygodny, prosty w użyciu i bezpieczny. Egzaminy można wypełniać nie ruszając się z domu. Nie ma konieczności ustalania terminów czy dojeżdżania na miejsce. Korzysta z autorskiego procesu automatyzacji procesów.

Pozwala na integrację z Systemami zarządzania nauczaniem (LMS), które są kompatybilne ze standardem LTI (*z ang. Learning Tools Interoperability*). Jest systemem agnostycznym i międzyplatformowym. Dzięki temu można z niego korzystać na najpopularniejszych przeglądarkach internetowych i systemach operacyjnych. Producent uważa, że jest narzędziem, które powoduje że egzaminy online są bezpieczne i pozbawione oszustw.

Dla studenta egzamin wygląda następująco:

- Logowanie do systemu zarządzania nauczaniem i wybór egzaminu,
- Pobranie i włączenie aplikacji Proctortrack,

- Weryfikacja tożsamości,
- Wypełnienie egzaminu.

Egzaminy, które są wspierane przez system, powinny być oznaczone jako współpracujące z narzędziem. Oczywiście program musi być pobrany i zainstalowany tylko raz. Aplikacja nie wymaga praw administratora systemu. Funkcja weryfikacji studenta odbywa się trzystopniowo. Należy pokazać swoją twarz, dowód tożsamości i skan kostek u dłoni. Pomiary odbywają sie za pomocą kamery internetowej i są kontrolowane na podstawie pliku, zawierającego dane biometryczne studenta. Po rozpoczęciu egzaminu, na ekranie pojawi się niebieskie obramowanie, symbolizujące zainicjowanie monitorowania zachowań. Po zakończeniu wypełniania testu oprogramowanie może być bezpiecznie odinstalowane. Wszystkie dane są wysyłane na zabezpieczone serwery, procesowane przez autorskie algorytmy, a wyniki dostarczane są bezpośrednio do wykładowców, w celu sprawdzenia.

Z perspektywy wykładowcy proces wygląda w następujący sposób:

- Włączenie wsparcia dla Proctortrack w systemie zarządzania nauczaniem,
- Konfiguracja funkcji nadzoru,
- Procesowanie egzaminów,
- Sprawdzanie wyników przez egzaminatora.

Tak jak z perspektywy studenta, cały proces rozpoczyna się w systemie zarządzania nauczaniem, gdzie odpowiedni egzamin powinien być oznaczony jak kompatybilny z narzędziem. Każdy egzaminator może dowolnie skonfigurować w jaki sposób będzie prowadzony nadzór podczas wypełniania egzaminu, zgodnie ze swoimi preferencjami. Każdy zakończony egzamin jest przepuszczany przez algorytmy, a wyniki są dostępne w ciągu 48 godzin. Studenci są monitorowani na podstawie zachowań, a każde budzące zastrzeżenia systemu jest zapisywane. Dzięki temu możliwe jest nadanie osobie wypełniającej egzamin oceny transparentności. Raporty są generowane dla każdego studenta i zawierają informacje takie jak:

- Skany związane z potwierdzeniem tożsamości,
- Zrzuty ekranu pulpitu wszystkich podejrzanych zachowań,
- Wideo z nagraniem wypełniania egzaminu, dla zachowania kontekstu.

System Proctortrack [35] oferowany przez firmę Verificient Technologies, to rozwiązanie zdecydowanie bardziej zaawansowane niż prototyp Legito, chociażby ze względu na jego komercjalizację i zespół osób, który za nim stoi. Działa on jednak na zupełnie innej zasadzie. Przede wszystkim jest to aplikacja pobierana na komputer, podczas gdy prototyp opisany w pracy to aplikacja internetowa. Praca narzędzia polega na nagrywaniu obrazu z kamery internetowej i pulpitu podczas wypełniania egzaminu. Legito nie oferuje takiej funkcjonalności, jednak znajduje się ona w propozycjach rozwoju w rozdziale 7.3. Procesowanie wyników odbywa się za pomocą algorytmów analizujących zachowanie osoby egzaminowanej. Sama myśl procesowania danych jest podobna w obu systemach. Należy jednak zauważyć, że wykorzystują w tym celu inne dane, a algorytmy w żadnym stopniu nie są ze sobą w podobne. Integracja, pomimo jej przewidzenia w prototypie, odbywa się w całkiem inny sposób. Prototyp ma umożliwiać użytkownikom logowanie do aplikacji na podstawie SSO (*z ang. Single Sign On*). Proctortrack jest stworzony w sposób

umożliwiający integrację na podstawie protokołu LTI, implementację stosowaną w rozwiązaniach E-Learning. Platforma ta nie ma możliwości rozszerzania jej funkcjonalności ze względu na zamknięty kod źródłowy i konieczność zapłaty za użytkowanie. Nie posiada funkcji tworzenia egzaminów, w tym celu należy użyć zewnętrznej platformy.

Typy zachowań wykrywane przez aplikację jako niepożądane:

- Używanie notatek,
- Wyjście z egzaminu,
- Zmiana osoby wypełniającej test,
- Pomoc innej osoby,
- Posiadanie uruchomionych aplikacji znajdujących się na czarnej liście,
- Kopiowanie pytań egzaminacyjnych.

Narzędzie można podsumować w prostych słowach. Generalizując, rozwiązuje ten sam problem, ale w zupełnie inny sposób. Wykorzystuje inne kryteria oceny w celu podjęcia decyzji czy student oszukuje.

2.4. Podsumowanie

Przedstawione w tym rozdziale rozwiązania zwalczają ten sam problem, starają się wyeliminować oszustwa podczas wypełniania egzaminów online. Robią to w zbliżony do siebie sposób, ponieważ każde z nich stawia na nadzór osoby podejmującej test.

Rozwiązaniem najmniej skalowalnym pod względem biznesowym oraz najmniej zaawansowanym technologicznie jest aplikacja Proctoru. Opiera się ona na połączeniu osoby egzaminowanej z nadzorcą w firmie. Rozwiązanie to jest ograniczone poprzez dostępność tych osób. Problem jest częściowo rozwiązany możliwością wyboru dogodnego dla egzaminowanego terminu, za pomocą kalendarza, który zawiera w sobie terminy dostępne z perspektywy nadzorujących. Proces ten nie jest zautomatyzowany, i jest to jego największa wada.

Najbardziej zaawansowanym technologicznie rozwiązaniem wydaje się być narzędzie AIProctor. Umożliwia ono korzystanie z możliwości wielu urządzeń jednocześnie, w celu monitorowania zachowań osoby egzaminowanej. Jego instalacja to po prostu dodanie wtyczki do przeglądarki Chrome, co jest dodatkowym ułatwieniem. Na podstawie nagrania z testu, algorytmy są w stanie znaleźć działania niepożądane i odpowiednio je oznaczyć. Integracja jest mocnym plusem tego rozwiązania - wyniki są wysyłane bezpośrednio do systemu zarządzania nauczaniem, z którego korzysta dana placówka.

Ostatnim systemem i tym, który wydaje się być złotym środkiem pomiędzy dwoma powyższymi, jest Proctortrack. Proces egzaminowania jest w pełni automatyczny. Posiada zaawansowane algorytmy, umożliwiające sprawdzenie czy egzaminowana jest właściwa osoba (skanowanie kostek u dłoni). Podobnie jak AIProctor, w rozwiązaniu tym osoba egzaminowana jest nagrywana kamerą internetową. Jest to jednak tylko jedno urządzenie - to podłączone do komputera. Pierwszoplanową rolę grają tu algorytmy sprawdzające niepoprawne zachowania.

W porównaniu do prototypu, każdy z tych systemów działa w inny sposób. Przede wszystkim korzystają one integracji z zewnętrznymi aplikacjami, umożliwiającymi tworzenie egzaminów. Legito jest natomiast aplikacją weryfikującą uczciwość na podstawie porównywania odpowiedzi pomiędzy sobą i skanowania Wikipedii.

Każde z tych narzędzi posiada inne funkcjonalności. Jednocześnie każde z nich jest też produktem komercyjnym, czyli za korzystanie z niego należy uiścić odpowiednią opłatę. Przy wyborze któregokolwiek czynnik ten powinien zostać wzięty pod uwagę.

3. Propozycja rozwiązania

W tym rozdziale przedstawione zostaną wymagania, które musi spełniać prototyp oraz wady i zalety egzaminowania online. We wstępnych założeniach aplikacja miała wyglądać inaczej niż produkt końcowy. Decyzja o zmianie została podjęta podczas pisania kodu. Autor uważa za ważne omówienie zmian względem pierwotnych założeń i przedstawienie toku myślenia, które towarzyszyły transformacji. Nazewnictwo komponentów aplikacji pozostało jak w wersji pierwszej. Ten rozdział powstał w celu lepszego zrozumienia działania systemu.

3.1. Wady i zalety egzaminowania online

W tym podrozdziale autor przedstawi jakie są plusy i minusy egzaminowania online, w porównaniu do przeprowadzania klasycznych egzaminów w salach wykładowych. Na samym początku warto przedstawić pozytywy:

- Oszczędność czasu,
- Oszczędność pieniędzy,
- Możliwość wykonania pracy zdalnie,
- Oszczędzanie środowiska,
- Uproszczony proces egzaminowania.

Oszczędność czasu można przełożyć na brak konieczności tworzenia testów w edytorach tekstu i ich późniejszego formatowania. Nie ma konieczności wydruku. Egzamin stworzony na platformie online, będzie sformatowany automatycznie. Oczywiście w dalszym ciągu musi nastąpić ocena odpowiedzi na pytania otwarte, jednak w przypadku zamkniętych (nawet tych wielokrotnego wyboru) dzieje się to automatycznie.

Oszczędność pieniędzy przejawia się w braku kosztów związanych z wydrukiem egzaminu. Nie potrzeba również wydawać pieniędzy na dojazd do miejsca przeprowadzania egzaminu.

Praca zdalna to brak konieczności dojazdu. Jest to chyba największy plus takiego rozwiązania. Studenci oraz egzaminator nie muszą stawić się w tym samym miejscu. Każda osoba może być w dowolnym miejscu na świecie. Jedyne, co jest potrzebne, to komputer i dostęp do Internetu.

Oszczędzanie środowiska to efekt uboczny całego procesu. Nikt nie musi dojechać do miejsca egzaminowania, co można rozumieć jako zmniejszenie emisji spalin. Nie trzeba wykorzystywać papieru do wydruku, dzięki czemu uratowane zostanie kilka drzew.

Poprzez uproszczony proces należy rozumieć to, że pewna część pracy zostaje zautomatyzowana. Ze strony egzaminatora wystarczy stworzyć szablon, w którym będą znajdować się pytania, i stworzyć listę studentów, którzy mają go wypełnić. Egzaminowani muszą wejść na odpowiednią stronę internetową i wypełnić test.

Niestety, w przypadku próby zastąpienia procesu, który dotychczas wymagał kontroli człowieka, w sytuacji kiedy proces ten ma działać prawidłowo, w przypadku egzaminowania online występują również pewne minusy, z którymi trzeba się pogodzić. Oto one:

- Podatność na oszustwa,
- Konieczność dostępu do komputera i Internetu.

Prototyp ma za zadanie redukować możliwości popełnienia oszustwa przez wypełniających egzamin na platformie online. Niestety, wyeliminowanie wszystkich prób jest zadaniem o bardzo wysokim poziomie skomplikowania, podobnie jak jego realizacja. Rozwijanie kolejnych funkcjonalności w prototypie, przedstawionych w rozdziałach 7.2 i 7.3, powinno przyczynić się do kolejnych redukcji potencjalnych możliwości oszustwa.

Konieczność dostępu do komputera i Internetu jest niezbędna. Jest to jeden z minusów, które nie mogą zostać wyeliminowane. Egzaminy muszą być przesyłane do systemu w celach weryfikacyjnych i nie da się tego zrealizować w inny sposób.

3.2. Wymagania

Dla zapewnienia poprawności działania prototyp powinien spełniać założenia, które są z nim związane. Wymagania są następujące:

- Zapewnienie rozszerzalności platformy,
- Możliwość integracji z istniejącymi systemami,
- Skalowalność,
- Możliwość tworzenia egzaminów,
- Możliwość konfiguracji zabezpieczeń podczas wypełniania egzaminów,
- Egzaminowanie.

Rozszerzalność powinna być zapewniona poprzez możliwość tworzenia i prostej integracji procesorów z systemem. W praktyce, oznacza to że dodanie kolejnego elementu skanującego egzaminy, odbywa się jak najmniejszym kosztem ze strony programistów, a ich czas można przeznaczyć na rozwój bardziej skomplikowanych procesorów.

Integracja z innymi systemami ma być możliwa poprzez udostępnienie usługi SSO (*z ang. Single Sign On*). Umożliwia ona rozszerzanie platformy poprzez dodawanie kolejnych aplikacji, mających wspólną bazę użytkowników. Możliwe jest też rozszerzenie jej w drugą stronę - użytkownicy innych aplikacji mogą posiadać dostęp do systemu, po jego wcześniejszej konfiguracji przez administratorów.

Skalowalność ma zostać zapewniona przez odpowiednią architekturę wybraną do stworzenia prototypu. Aplikacja musi być wysoko dostępna i minimalizować ryzyko wystąpienia utraty danych.

Możliwość tworzenia egzaminów musi być dostępna w aplikacji klienckiej. Dodawanie pytań ma odbywać się dynamicznie - każdy egzamin może mieć różną ich ilość. Każde pytanie posiada wybór typu: otwarte lub zamknięte. Dla pytań zamkniętych dostępna jest możliwość wskazania poprawnej odpowiedzi. Egzaminator musi mieć możliwość dodawania konkretnych osób mogących wypełnić test, bazując na ich loginie lub adresie e-mail.

Konfiguracja zabezpieczeń podczas wypełniania egzaminu powinna być dostępna dla każdego egzaminatora indywidualnie. Może on wybrać, które zabezpieczenia mają być włączone, a które wyłączone.

Egzaminowanie jest dostępne na podstawie danych otrzymanych z systemu. Pytania generują się w sposób automatyczny i adekwatny do ich typu. Funkcje zabezpieczające działają w sposób zadeklarowany przez egzaminatora.

3.3. Wstępna koncepcja

Kluczowym elementem systemu Legito miała być biblioteka, która służyłaby do implementacji w systemach zewnętrznych. Czyli istniałaby możliwość podpięcia systemu do istniejących rozwiązań e-learningowych lub egzaminujących. Elementy systemu to:

- Biblioteka dla zewnętrznych klientów,
- API `Legito.Api`,
- API `Legito.BackOffice.Api`,
- API `Legito.ProcessorCoordinator.Api`,
- Serwer autoryzacji i autentykacji `Legito.AuthorizationServer`,
- Aplikacja kliencka `BackOffice` służąca do zarządzania,
- Odkrywanie usług,
- Kolejka komunikatów,
- Serwer przechowujący logi aplikacji i Kibana,
- Procesory.

Zaczynając od początku, `Legito.Api` to aplikacja, która jest odpowiedzialna za składowanie rozwiązanych egzaminów i ich wzorów, korzystająca z bazy danych NoSQL RavenDB. Jest to punkt, z którego miała korzystać biblioteka kliencka.

Drugie z kolei, `Legito.BackOffice.Api`, służy do zarządzania aplikacją przez wykładowców. Miało się tam odbywać tworzenie egzaminów, konfiguracja elementów zapobiegających oszustwom, rozpoczęcie i zakończenie egzaminu, start procesowania rozwiązanych testów i wyświetlanie wyników. API korzysta z relacyjnej bazy danych SQL Server.

`Legito.ProcessorCoordinator.Api` to swojego rodzaju serce systemu. Jest to aplikacja, która odpowiada za komunikację z procesorami egzaminów i koordynuję ich pracę. Korzysta z relacyjnej bazy danych SQL Server.

`Legito.AuthorizationServer` miał zabezpieczać poszczególne elementy systemu w komunikacji między sobą i autentykować użytkowników aplikacji `BackOffice`.

Klienci, czyli egzaminatorzy, mieli korzystać z aplikacji `BackOffice`, w celu tworzenia egzaminów i sprawowania nad nimi kontroli za pomocą ustawień zabezpieczających przez oszustwami. Wykreowane wzory egzaminów i konfiguracje miały trafiać do zintegrowanych systemów dzięki bibliotece.

Odkrywanie usług odpowiada za przechowywanie informacji o poszczególnych elementach systemu i kierowaniu ruchu do odpowiednich aplikacji.

Kolejka komunikatów to asynchroniczny serwer, odpowiadający za komunikację pomiędzy `Legito.ProcessorCoordinator.Api`, a procesorami egzaminów.

Serwer przechowujący logi aplikacji to centralne miejsce, w którym są zapisywane błędy i komunikaty z aplikacji API, występujących w systemie. Połączony jest z aplikacją do obsługi bazy Kibana.

Procesory to niezależne aplikacje, do których trafiają rozwiązane paczki, zawierające wypełnione egzaminy. Wykonują one na nich operacje sprawdzające czy zaszło oszustwo i przesyłają wyniki do `Legito.ProcessorCoordinator.Api`.

3.4. Finalna koncepcja

W finalnej wersji prototypu idea biblioteki, służącej do zewnętrznych integracji z systemami egzaminującymi i e-learning, została porzucona. Przyczyna była zupełnie prosta. Konstruując aplikację, która umożliwia tworzenie, sprawdzanie i weryfikację egzaminów, dopisanie fragmentu kodu egzaminującego studentów jest dużo prostszym rozwiązaniem od biblioteki. W ten sposób platforma jest bardziej spójna, a integracja jest dalej możliwa.

Elementy systemu to:

- API `Legito.Api`,
- API `Legito.BackOffice.Api`,
- API `Legito.ProcessorCoordinator.Api`,
- Serwer autoryzacji i autentykacji `Legito.AuthorizationServer`,
- Aplikacja kliencka `BackOffice`,
- Odkrywanie usług,
- Kolejka komunikatów,
- Serwer przechowujący logi aplikacji i Kibana,
- Procesory.

`Legito.Api` pozostaje w swojej nie zmienionej formie. W dalszym ciągu jest to mikroserwis odpowiadający za składowanie egzaminów i wzorów egzaminów oraz umożliwiający operacje związane z nimi.

`Legito.BackOffice.Api`, w porównaniu do pierwotnej koncepcji, otrzymało dodatkowe zadanie i stało się bramą federacyjną (z ang. *federation gateway*) dla aplikacji klienckiej. Jest to mikroserwis, który służy za swojego rodzaju proxy pomiędzy klientem, a poszczególnymi elementami systemu.

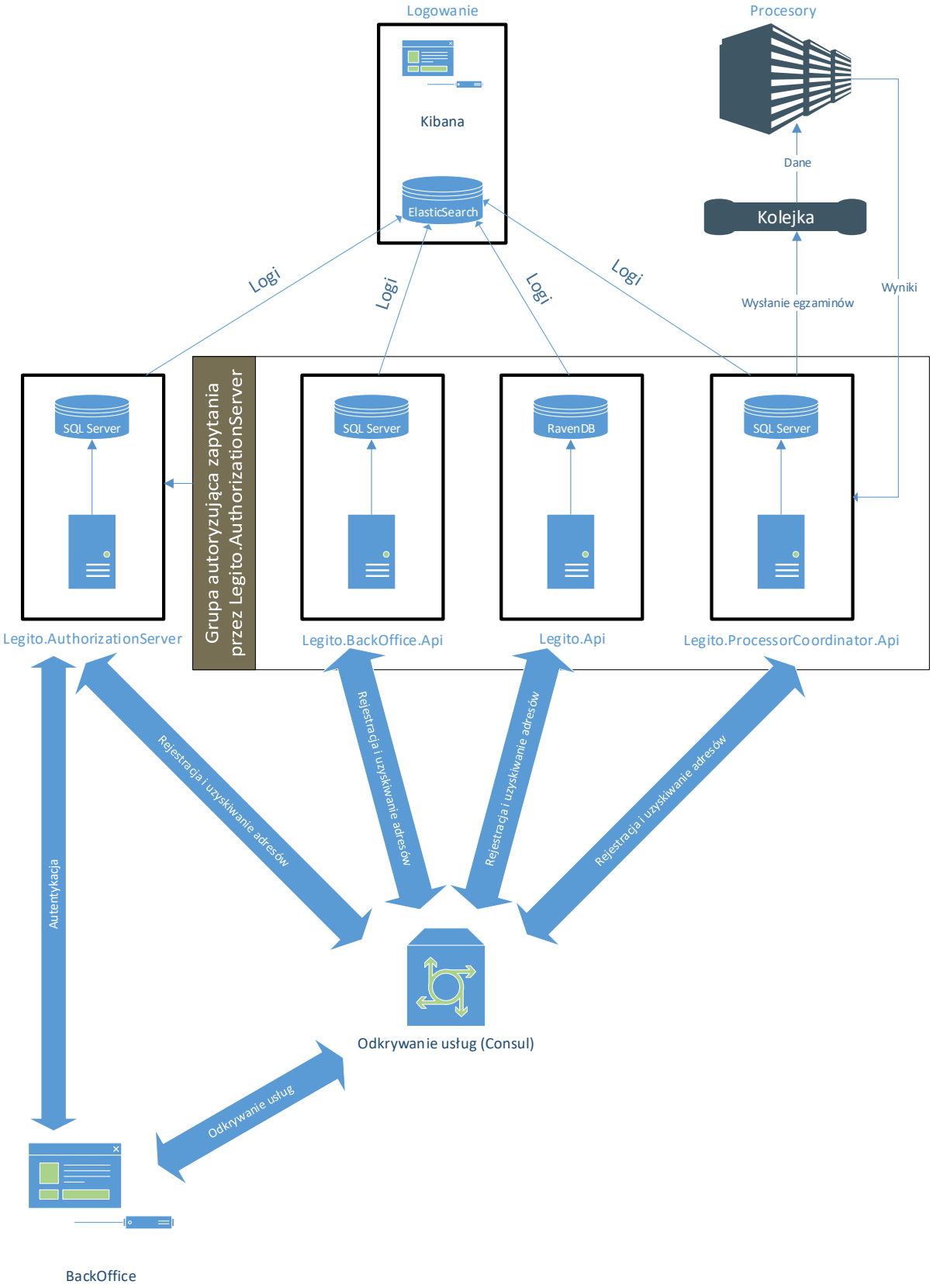
Rola `Legito.ProcessorCoordinator.Api` pozostała bez zmian. W dalszym ciągu jest to swoiste serce systemu koordynujące pracę procesorów skanujących egzaminy i udostępniające dane wynikowe. Posiada logikę odpowiadającą za dodawanie kolejnych procesorów.

Serwer autentykacji i autoryzacji również pełni taką samą rolę jak w pierwotnej koncepcji. Umożliwia bezpieczną komunikację pomiędzy usługami, umożliwia logowanie użytkowników i przechowuje wszystkie dane związane z ich kontami. Dzięki niemu możliwa jest integracja z zewnętrznymi systemami używając tzw. SSO (z ang. Single Sign On).

Aplikacja `BackOffice` dostała dodatkowe funkcjonalności związane z egzaminowaniem studentów. Umożliwia ona nie tylko konfigurację ustawień testowych, tworzenie egzaminów, skanowanie ich i wyświetlanie wyników, ale również posiada interfejs przeznaczony tylko dla osób posiadających rolę ucznia.

Pozostałe elementy systemu pełnią dokładnie takie same funkcje, jakie były założone w pierwszej koncepcji. Jedyną zmianą, przy porównaniu tych dwóch idei to decyzja o miejscu podejmowania egzaminów. Implementacja jej w aplikacji klienckiej była dużo lepszym rozwiązaniem, patrząc z perspektywy całego systemu. Przy ewentualnych aktualizacjach, cała logika pozostaje w jednym miejscu. Zewnętrzni administratorzy nie muszą pobierać żadnej nowej wersji biblioteki ani poznawać jej tajników. Integracja i rozszerzalność, czyli najważniejsze założenia, są dalej spełnione.

Architektura systemu jest przedstawiona na rysunku 1. Poszczególne elementy zostaną opisane w rozdziale 5. Warto w tym miejscu zaznaczyć, że rysunek 1 nie przedstawia pełnej komunikacji pomiędzy elementami. Należy założyć, że wszystkie aplikacje, oprócz `BackOffice`, które komunikują się z odkrywaniem usług, mogą się między sobą komunikować.



Rysunek 1. Architektura systemu

Źródło: Opracowanie własne

4. Narzędzia i technologie wykorzystane w pracy

Rozdział ten został podzielony na sześć części. Pierwsza dotyczy języków programowania, użytych do stworzenia prototypu. Druga zawiera opis Frameworków. Trzecia, to biblioteki zastosowane w pracy. Czwarta, prezentuje narzędzia które zostały użyte w tworzeniu systemu. Piąta przybliża użyte typy baz danych. Ostatnia, szósta, opisuje rozwiązania użyte do wirtualizacji i komunikacji.

4.1. Języki programowania

W podrozdziale wymienione zostały języki programowania, pojawiające sie w prototypie. Każdy z pod-podrozdziałów zawiera opis technologii, jej cechy, krótką historię i motywację do wykorzystania przy tworzeniu prototypu.

4.1.1 C#

C# [1] jest językiem obiektowym, bezpiecznie typowanym (*z ang. type-safe*). Służy do budowy systemów różnego typu, wykorzystując .NET Framework. Można dzięki niemu tworzyć m.in. aplikacje internetowe (ASP.NET, ASP.NET MVC, .NET Core), desktopowe (WinForms, WPF), elementy systemów rozproszonych (WCF) i międzyplatformowe programy na smartfony przy użyciu Xamarin.

Jego składnia jest przejrzysta i łatwa do nauczenia. Powinien być łatwo przyswajalny dla osób, które wcześniej pracowały z językami takimi jak Java, C czy C++.

Wspiera tworzenie generycznych metod i typów, które przyczyniają się do zwiększenia bezpiecznego typowania i wydajności tworzonego oprogramowania. Ważnym aspektem, o którym warto wspomnieć, jest możliwość pisania wyrażeń LINQ (*z ang. Language-Integrated Query*). Umożliwiają one dostęp do interesujących nas danych, używając prostej i czytelnej składni. Można ich używać m.in. do odpytywania bazy danych, czy przeszukiwania kolekcji.

C# [2] został stworzony przez Andersa Hejlsberga i jego zespół w korporacji Microsoft, przez którą jest dalej rozwijany i utrzymywany. Jego twórca i główny architekt jest postacią nadzwyczaj ciekawą - wcześniej współtworzył Turbo Pascala i Embarcadero Delphi. Pierwsza wersja języka została publicznie ogłoszona na konferencji Professional Developers Conference w lipcu 2000 roku.

Język ten został wybrany do stworzenia prototypu ze względu na jego prostą składnię, skrócony czas prac programistycznych i doświadczenie autora w pracy z nim. Współpracując z Frameworkiem .NET Core, język ten ma dużą wydajność, pracując na maszynach z systemem operacyjnym Linux, co było ważnym aspektem przy użyciu wirtualizacji w postaci kontenerów Dockerowych, o których więcej zostało napisane w podrozdziale 4.6.1.

4.1.2 TypeScript

TypeScript [3][4] jest językiem stworzonym i utrzymywanym przez korporację Microsoft. Jego autorem jest znany z rozdziału 4.1.2 Anders Hejlsberg. Dystrybucja odbywa się na podstawie licencji Apache 2.0. TypeScript jest nadzbiorem (*z ang. superset*) JavaScript - kompiluje się do

tego właśnie języka, dzięki czemu jest wieloplatformowy. Zawiera wszystkie jego funkcjonalności, a nawet rozszerza go o kolejne. W przeciwieństwie do swojego prekursora, jest językiem silnie typowanym, co jest sprawdzane na etapie kompilacji.

Powstało wiele natywnych Frameworków współpracujących z tym językiem takich jak:

- Angular,
- Ionic,
- RxJs5,
- Dojo 2.

Dzięki jego pokrewieństwu z JavaScript jest on w stanie wykorzystywać biblioteki napisane w nim, takie jak jQuery, Bootstrap czy Underscore. Są to jedynie przykłady, gdyż można użyć praktycznie każdej biblioteki. Działa to oczywiście w dwie strony - skompilowany TypeScript może zostać wykorzystany w programach napisanych w JavaScript.

Został stworzony ze względu na zwiększenie się stopnia skomplikowania aplikacji pisanych w języku JavaScript. Dlatego skupiono się na tym, żeby kod pisany w nim mógł być ponownie użyty. Implementuje wiele rozwiązań znanych z języków zorientowanych obiektowo:

- Statyczne typowanie,
- Klasy,
- Interfejsy,
- Generyczność,
- Moduły.

Motywacją do wykorzystania go w pracy jest jego powszechność i dostępność dokumentacji. Dla programistów związanych z technologiami takimi jak C# czy Java, jego składnia jest bardzo czytelna, łatwa do zrozumienia.

4.1.3 JavaScript

JavaScript [5] jest skryptowym językiem programowania. Został stworzony przez Brendana Eicha w 1995 roku. W wczesnych fazach rozwoju język był poddawany krytyce ze względu na brak planowania naprzód podczas jego implementacji. Można jednak powiedzieć, że JavaScript wyprzedził swoją epokę. Współczesnym programistom zajęło piętnaście lat, żeby zrozumieć stopień wyrafinowania jaki rządzi jego światem.

Początkowo język ten był nazywany Mocha, później przez chwilę LiveScript, żeby skończyć jako JavaScript. Nazewnictwo nie jest tutaj przypadkowe, jako że Java była wtedy popularnym językiem programowania.

W listopadzie 1996 roku firma Netscape zatwierdziła język do Ecma, prywatnej organizacji non profit ,w celu jego standaryzacji. Stworzyli oni specyfikację ECMA-26, która jest istotą języka JavaScript. Technicznie rzecz biorąc, JavaScript jest implementacją ECMAScript, czyli specyfikacji firmy Ecma. Nazwy te mogą być stosowane wymiennie.

Wykorzystywany jest głównie do tworzenia tego, co widać w aplikacjach internetowych, czyli warstwy użytkownika. Jest wieloplatformowy, dzięki czemu działa poprawnie w wielu

przeglądarkach internetowych. Może być wykorzystywany do budowy warstwy serwerowej przy użyciu serwera Node.js, czy do tworzenia aplikacji na smart fony dzięki Apache Cordova.

W pracy został użyty przy współpracy z TypeScript, najczęściej w postaci bibliotek jak np. jQuery. Wybór był tak naprawdę mimowolny - niektóre fragmenty muszą zostać napisane w JavaScript, pomimo korzystania z TypeScript.

4.1.4 SQL

SQL [6] (*z ang. Structured Query Language*) to język, który umożliwia pracę z bazami danych. Jest standardem wpisanym do ANSI (American National Standards Institute), jednak istnieją różne jego odmiany jak T-SQL czy PL/SQL. Współpracuje z relacyjnymi bazami danych takimi jak:

- SQL Server,
- MySQL,
- Oracle,
- Postgres,
- Informix.

Wszystkie z nich używają go jako podstawowego języka do obsługi danych. Wersja ta sprawdza się świetnie jeśli jednym z wymagań systemowych jest używanie wielu baz.

W roku 1970 Dr. Edgar Codd opisał relacyjny model baz danych. Następnie w 1974 pojawił się Structured Query Language. Całe dwanaście lat później w 1986 roku, IBM stworzył prototyp relacyjnej bazy danych opartej o standard ANSI. Pierwsza tego typu baza została wydana przez Relational Software, znana później jako Oracle.

SQL w prototypie używany jest w celu dostępu do bazy danych, którą jest SQL Server 2017. Nie jest on jednak używany bezpośrednio, a generowany przez Entity Framework w celu komunikacji. Język ten został tak naprawdę narzucony wraz z wyborem technologii relacyjnej bazy danych. Jest to typowy wybór przy używaniu stosu technologicznego od firmy Microsoft.

4.2. Frameworki

W tym podrozdziale autor przedstawia Frameworki wykorzystane do stworzenia prototypu. W przeciwieństwie do poprzedniego podrozdziału, poświęconego językom programowania, tutaj nie zostanie przytoczona ich dłuższa historia. Jest to spowodowane tym, że są to na tę chwilę rozwiązania świeże. Opisane zostanie ich zastosowanie, działanie i motywacja do użycia w prototypie systemu.

4.2.1 ASP.NET Core 2.1

ASP.NET Core [7] jest Frameworkiem wieloplatformowym i wysoko wydajnościowym. Propagowany przez open source, jest używany do budowy nowoczesnych systemów w oparciu o rozwiązania chmurowe. Wspierany i utrzymywany przez korporację Microsoft. Dzięki niemu możliwa jest budowa m.in. aplikacji internetowych i IoT (*z ang. Internet of Things*). Jest on

przeprojektowanym Frameworkiem ASP.NET 4.x z pewnymi zmianami architekturalnymi, co skutkuje jego odchudzeniem i większą modularnością.

Używając go, otrzymuje się następujące korzyści:

- Integrację z nowoczesnymi Frameworkami klienckimi jak np. Angular,
- Przygotowanie pod chmurę z konfiguracjami zależnymi od środowiska,
- Wbudowane wstrzykiwanie zależności,
- Lekki, wysoko wydajnościowy, modularny potok (*z ang. pipeline*) dla zapytań HTTP,
- Możliwość hostowania z użyciem IIS, Nginx, Apache, Dockera lub we własnym procesie,
- Narzędzia ułatwiające nowoczesne tworzenie aplikacji internetowych,
- Możliwość działania na Windows, Linux i maxOS.

Jest rozpowszechniany w całości przy użyciu paczek NuGet, dzięki czemu programista ma możliwość używania tylko niezbędnych w danym projekcie bibliotek, co skutkuje odchudzeniem projektu. Oprócz zysku na rozmiarze, otrzymuje się również zwiększone bezpieczeństwo aplikacji i wydajność.

Firma Microsoft cały czas dąży do standaryzowania, czego rezultatem jest przeniesienie znacznej części API tzw. pełnego Frameworka .NET do .NET Core.

Motywem Użycia tego właśnie Frameworka przy budowie prototypu, była potrzeba stworzenia systemu w architekturze mikroserwisów. Jest to rekomendowana droga z użyciem technologii Microsoft. Dzięki możliwości hostowania aplikacji w kontenerach Docker można stworzyć system skalowalny i wysoko dostępny.

4.2.2 Angular

Angular [8][9] jest drugim wcieleniem popularnego Frameworka AngularjJS. Został dostosowany do najnowszych standardów tworzenia aplikacji internetowych. Posiada pełne wsparcie dla ES6 i TypeScript. Aplikacje tworzone w nim wykorzystują dobrze znany wzorzec MVC (*z ang. Model-View-Controller*). Dzięki niemu są:

- Rozszerzalne, znając podstawy działania Frameworka łatwo jest się odnaleźć w kodzie,
- Łatwe w utrzymaniu i naprawie ewentualnych błędów, dzięki możliwości debugowania,
- Testowalne, posiada wsparcie bibliotek zapewniających możliwość testowania, nie tylko jednostkowego, ale również *end-to-end,*
- Jest standardem, który wykorzystuje możliwości przeglądarki internetowej oraz popularnych narzędzi i bibliotek.

Można powiedzieć, że pisanie w nim jest proste i przyjemne.

Od strony bardziej technicznej, posiada wbudowane wstrzykiwanie zależności i własny routing. Dzięki temu pozwala na budowę aplikacji modularnych, a ich kod jest

samodokumentujący. W swojej drugiej odsłonie Angular jest nawet pięciokrotnie szybszy od swojego poprzednika.

Został stworzony w korporacji Google, przez którą jest dalej utrzymywany i stale udoskonalany. W porównaniu z AngularJS, nowsza wersja jest bardziej rozbudowana i może być przytłaczająca dla osób, które chcą sie jej nauczyć. Wcześniejsza styczność z wersją pierwszą na niewiele się zda - można wręcz powiedzieć, że jest to zupełnie inny produkt.

Decyzja o użyciu go w pracy została podjęta stosunkowo późno, co zostanie dokładnie wyjaśnione w dalszej części pracy. Służy on do utylizacji API zarządzającego aplikacją. Dzięki jego znajomości autor miał ułatwione zadanie w tworzeniu warstwy użytkownika. Wzorzec MVC i składnia zbliżona do C# zapewnił szybkie stworzenie tego komponentu systemu.

4.3. Biblioteki

W tym rozdziale autor przedstawia zewnętrzne biblioteki, które zostały użyte do stworzenia prototypu. Opisane jest ich działanie i zastosowanie w pracy.

4.3.1 IdentityServer4

IdentityServer4 [12] to biblioteka, która implementuje OpenID Connect [10] i OAuth 2.0 [11]. Dzięki niej można uzyskać w aplikacji:

- Autentykację jako serwis,
- Sigle Sign-on,
- Kontrolę dostępu do API,
- Bramę federacyjną (*z ang. Federation Gateway*).

Twórcy skupili się na jak największej możliwości konfiguracji i rozszerzalności swojego rozwiązania. Jest to dojrzały open source, rozpowszechniany na podstawie licencji Apache 2.

OpenID Connect jest to warstwa abstrakcji nad protokołem OAuth 2.0. Umożliwia użytkownikom potwierdzenie swojej tożsamości i otrzymanie podstawowych danych profilu utylizując w tym celu endpointy REST. Jest to pewnego rodzaju standard, umożliwiający różnym aplikacjom klienckim, w tym internetowym i na smartfony, wysyłanie zapytań i odbieranie informacji o autentykacji sesji. Protokół jest rozszerzalny i daje możliwość używania opcjonalnych funkcji, takich jak szyfrowanie danych czy zarządzanie sesją.

OAuth 2.0 jest standardowym protokołem wykorzystywanym do autoryzacji. Skupia się na jak największym uproszczeniu implementacji i jednocześnie oferuje wiele możliwości konfiguracji potoku (*z ang. flow*) autoryzacji w aplikacjach internetowych i nie tylko. Jego specyfikacja i rozszerzenia są stale rozwijane przez IETF OAuth Working Group.

Autentykacja jest potrzebna w momencie, kiedy niezbędna jest wiedza o tym, kto w danym momencie używa aplikacji. Autoryzacja natomiast służy do identyfikacji i nadawania dostępu pomiędzy aplikacjami.

W prototypie biblioteka została użyta ze względu na konieczność zabezpieczenia systemu. Dzięki niej i jej wdrożeniu jako osobne API, powstał osobny komponent do zarządzania wszystkim

co związane z użytkownikami i dostępem. Dzięki implementacji wyżej wymienionych protokołów nie ma problemu z rozszerzaniem systemu. Można również zaimplementować w kilku liniach kodu integracje z Facebook i Google.

4.3.2 Entity Framework Core

Entity Framework Core [13] to lekka wieloplatformowa wersja popularnego ORM Entity Framework. Służy do uzyskaniu dostępu do bazy danych i wykonywania na niej operacji. Wspiera bazy danych takie jak SQL Server, MySQL i inne przy zastosowaniu odpowiednich rozszerzeń.

Zawiera wiele funkcji typowych dla ORM [14]:

- Mapowanie obiektów POCO (*z ang. plain old c# object*),
- Eager i Lazy loading,
- Tłumaczenie zapytań LINQ na SQL,
- Mapowania jeden do jednego, jeden do wielu i wiele do wielu,
- Strategie dziedziczenia,
- Typy złożone,
- Podejście *code-first*,
- Migracje.

Został stworzony przez korporację Microsoft, przez którą jest dalej utrzymywany. Jego kod jest dostępny jako open source na GitHub.

Został wykorzystany w pracy ze względu na to, że w momencie rozpoczęcia pisania kodu był to jedyny pełny ORM dostępny na platformę .NET Core. Nie zmienia to jednak faktu, że praca z nim, dzięki wykorzystaniu migracji i podejścia *code-first*, należy do bardzo przyjemnych. Jeśli rozpoczęcie pracy nad kodem przedłużyłoby się o kilka miesięcy, autor wziąłby pod uwagę możliwość użycia nHibernate.

4.3.3 Autofac

Autofac [15] jest kontenerem IoC (z ang. Inversion of Control). Jest to projekt open source, rozpowszechniany na podstawie licencji MIT, więc może być swobodnie wykorzystywany w projektach komercyjnych.

Służy do wstrzykiwania zależności (*z ang. dependency injection*) w projekcie. Można go bardzo szeroko konfigurować - w zależności od wymagań projektowych. Jest podzielony na moduły, dzięki czemu czytelna struktura kodu pozostaje zachowana. Wykorzystywany głównie w aplikacjach internetowych i komponentach związanych z architekturą rozproszoną. Skupia się na wokół .NET Framework.

W prototypie użyty ze względu na szybkość jego działania. Możliwość tworzenia modułów do konfiguracji rozwiązywania zależności dla poszczególnych warstw aplikacji. Łatwość użycia - jego API jest bardzo czytelne i proste do nauki. Komplikacje pojawiają się dopiero przy bardzo zaawansowanych scenariuszach.

4.3.4 AutoMapper

AutoMapper [16][17] jest małą biblioteką służącą do mapowania obiektów na inne obiekty. Działa na zasadzie przyjęcia obiektu na wejściu i zwróceniu obiektu innego typu. Cechą wyróżniającą tę bibliotekę to możliwość konfiguracji. Można wskazywać, która właściwość ma być przekształcona w którą, lub wykonywać operację po zmapowaniu np. kolekcji. Jeśli obiekty zawierają pola o takich samych nazwach, konfiguracja jest minimalna - nie trzeba ich deklarować.

Wykorzystywany jest głównie przy przekształcaniu obiektów pomiędzy warstwami aplikacji jak DTO (*z ang. Data Transfer Object*) na encje bazy danych. Dzięki niemu nie trzeba tworzyć nowego obiektu i przekazywać mu wartości - biblioteka robi to za nas i to w jednej linii kodu!

Jego użycie jest nadzwyczaj proste. Wystarczy zainicjalizować obiekt mapujący i wczytać do niego zadeklarowaną konfigurację.

Jest to projekt open source dystrybuowany na licencji MIT. Został stworzony przez Jimmiego Bogarda i innych współautorów. Jest wspierany przez .NET Foundation.

W pracy został użyty w celu mapowania obiektów pomiędzy warstwami aplikacji. Głównie w warstwie komunikacji i bazy danych. Jest to najpopularniejsza biblioteka tego typu.

4.3.5 Serilog

Serilog [18] to biblioteka diagnostyczna, pozwalająca na logowanie w aplikacjach na platformie .NET Core. Jest prosta w użytkowaniu i posiada łatwe API, służące do jej obsługi. Nadaje się do współpracy zarówno z prostymi, jak i ze skomplikowanymi aplikacjami. Jej funkcjonalności to m.in:

- Posiada wsparcie społeczności,
- Aktywnie rozwijana,
- Formatowanie na podstawie poziomów logowania,
- Najlepsze wsparcie dla .NET Core,
- Wiele opcji przechowywania/zapisu logów.

Jest projektem open source, wpieranym i rozwijanym przez społeczność. Dystrybuowana na podstawie licencji Apache 2.0.

W projekcie wykorzystana ze względu na prostotę użycia. Konfiguracja to dosłownie parę linijek kodu. Używana jest wraz bazą ElasticSearch i narzędziem do wyświetlania danych Kibana. Razem tworzą świetny zestaw do monitorowania stanu aplikacji.

4.3.6 xUnit.net

xUnit.net jest narzędziem do testowania aplikacji pisanych z użyciem .NET Framework. Jest to biblioteka darmowa, open source, zorientowana na społeczność przez którą jest utrzymywana i rozwijana. Jest częścią .NET Foundation i operuje na podstawie *code of conduct*. Rozpowszechniana na podstawie licencji Apache 2.

Napisana przez autora dobrze znanej biblioteki NUnit v2. Dzięki temu jej użycie jest znajome i proste.

Wybrana do prototypu na podstawie znajomości autora jej wcześniejszej wersji na pełny Framework .NET. Jej użycie jest łatwe, a integracja z narzędziami do budowania i testowania projektu nie sprawia problemów.

4.4. Narzędzia

W tym rozdziale zostaną opisane wszystkie narzędzia, które zostały wykorzystane w budowie prototypu. Znajdują się tu nie tylko IDE, ale także elementy wspomagające pisanie kodu, czy też wykorzystywane w działaniu platformy jako całości.

4.4.1 Visual Studio 2017

Visual Studio 2017 [19] to zintegrowane środowisko programistyczne, wykorzystywane do tworzenia aplikacji wykorzystując język C# i inne. Zostało stworzone przez korporację Microsoft i jest przez nią utrzymywane i rozwijane. Rozpowszechniane na podstawie licencji komercyjnych o zróżnicowanych funkcjach i cenach. Istnieje również wersja bezpłatna, która może być używana przez małe firmy. Udoskonalenia:

- Ułatwienia do wprowadzenia metodologii DevOps,
- Poprawiona nawigacja,
- Jeszcze lepsze IntelliSense,
- Więcej możliwości refaktoryzacji.

Dzięki zmianom w procesie debugowania udało się znacząco uprościć znajdowanie i rozwiązywanie problemów z aplikacją. W ten sposób ograniczono m.in. możliwość regresji podczas prac programistycznych.

Visual Studio jest w pełni zintegrowane z chmurą Azure. Posiada prawie jednoklikowy proces wysyłania aplikacji na serwer (*ang. deployment*). Współpracuje m.in. z Azure Functions, które jest jedną z nowości chmurowych od firmy Microsoft. Wpiera kontenery Docker, system Windows i inne.

Oferuje zarządzanie projektami we współpracy z VSTS (*z ang. Visual Studio Team Services*) czy TFS (*z ang. Team Foundation Server*). Jedną z nowości jest możliwość otwarcia dowolnego folderu z plikami z kodem i natychmiastowe rozpoczęcie pracy z nimi.

Najnowsza wersja Visual Studio posiada ulepszony instalator i sam proces instalacji. Oferuje on możliwość wyboru poszczególnych komponentów, które mają zostać zainstalowane. Zawarto również wiele udoskonaleń, które mają wpływ na pracę:

- Krótszy czas uruchamiana,
- Odciążenie pamięci,
- Zredukowany czas odpowiedzi.

W tworzeniu prototypu Visual Studio zostało wykorzystane do napisania całej warstwy serwerowej. Jego pełna współpraca z .NET Core i Docker daje świetne wyniki, a IntelliSense znacznie przyśpiesza pisanie poprawnego kodu. Jest to domyślny wybór przy tworzeniu aplikacji z wykorzystaniem języka C#.s

4.4.2 ReSharper

ReSharper [20] to wtyczka do Visual Studio, zwiększająca produktywność podczas pisania kodu. Posiada rozbudowany system automatyzacji, co znacznie przyśpiesza pracę. Pomaga odnaleźć w kodzie:

- Błędy kompilatora,
- Błędy środowiska uruchomieniowego,
- Redundancję,
- "Śmierdzący kod".

Co więcej, oferuje inteligentne rozwiązania powyższych już w trakcie pisania. Pomaga również w nawigacji po kodzie, dzięki wizualizacjom struktur i hierarchii.

Dzięki funkcjom refaktoryzacji, pozwala robić zmiany na poziomie całej solucji, bez zmartwień o to, że kod przestanie się kompilować. Wbudowane formatowanie kodu i jego sprzątanie pozwala na zachowanie standardów pisania kodu w całym zespole programistycznym.

Został stworzony przez firmę JetBrains, która zajmuje się jego dalszym rozwojem i utrzymaniem. Jest to produkt komercyjny, jednak istnieją licencje, które pozwalają na korzystanie z narzędzia studentom szkół wyższych w ramach nauki.

W pracy został użyty do zachowania wysokiego standardu kodu, ale w głównej mierze do sprawniejszego pisania kodu, wykorzystując wiele z jego możliwości.

4.4.3 WebStorm

WebStorm [21] to podobnie jak Visual Studio zintegrowane środowisko programistyczne. Jednak, w przeciwieństwie do niego, skupia się na języku JavaScript. Nie bez przyczyny reklamowany jako najsprytniejszy edytor. Jego producentem jest ta sama firma, która wyprodukowała wtyczkę ReSharper opisaną wyżej. Oferuje:

- Inteligentne podpowiadanie kodu w trakcie pisania,
- Detekcję błędów w locie,
- Zaawansowaną nawigację,
- Funkcję refaktoringu,
- Wsparcie dla języków JavaScript, TypeScript, styli (jak CSS) i popularnych frameworków jak Angular czy React.

Dzięki funkcji debugowania można pracować w podobny sposób jak z aplikacjami na warstwie serwerowej. Posiada wsparcie dla różnych wtyczek, które rozszerzają jego działanie. Wspiera pisanie testów jednostkowych, używając do tego popularnych bibliotek, takich jak Karma. Dzięki wbudowanej możliwości współpracy z systemami kontroli wersji można w prosty sposób zintegrować się np. z systemem GIT.

Jest to produkt rozpowszechniany na podstawie licencji komercyjnej. Istnieje jednak jego darmowa wersja jak w przypadku ReSharper, skierowana do studentów.

W projekcie użyty do napisania całej warstwy użytkownika, wykorzystując Framework Angular i język TypeScript. Narzędzie okazało się dużo sprawniejsze od Visual Studio Code, dlatego wybór padł właśnie na nie.

4.4.4 Consul

Consul [22] jest rozwiązaniem oferującym siatkę usług (*z ang. service mesh*) i odpowiada za komunikację pomiędzy mikroserwisami. Może służyć m.in. jako aplikacja dostarczająca odkrywanie usług. Każda z jego funkcjonalności może być używana osobno. Podsiada wbudowane proste Proxy, dzięki czemu może działać zaraz po zainstalowaniu. Jego główne funkcje to:

- Odkrywanie usług, klienci mogą się rejestrować i odpytywać o adresy innych usług poprzez HTTP lub DNS,
- Sprawdzanie zdrowia (*z ang. health check*). Konfiguracja umożliwia deklarację endpointa pod którym aplikacja będzie odpytywana o jej stan,
- Baza klucz - wartość (*z ang. KV Store*),
- Bezpieczna komunikacja między usługami, dzięki możliości generowania i dystrybuowania certyfikatów CLS.

Został zaprojektowany jako narzędzie, które ma być przyjazne podejściu DevOps i programistom. Idealnie dopasowane do nowoczesnych rozwiązań chmurowych, gdzie aplikacje muszą być elastyczne.

Consul został stworzony przez firmę HashiCorp. Jest to projekt open source, więc jest utrzymywany zarówno przez firmę i społeczność. Posiada darmową licencję, jak również komercyjną do działania przy rozwiązaniach klasy korporacyjnej.

W prototypie użyty do odkrywania usług i jako punkt, który sprawdza stan zdrowia poszczególnych mikroserwisów. Zastosowany w projekcie ze względu na swoje szerokie możliwości, z myślą o przyszłym rozwoju aplikacji, korzystając z innych jego funkcji.

4.4.5 Kibana

Kibana [23] to narzędzie pozwalające wizualizować i nawigować dane z bazy ElasticSearch. Wprowadza dużo możliwości grupowania i badania wyników zapytań dla lepszego zrozumienia działania aplikacji i znalezienia pewnych trendów. Dzięki interaktywnym grafom pozwala w prosty sposób wizualnie zlokalizować interesujące wartości.

W swojej podstawowej wersji dostarczana z histogramami, diagramami kołowymi i wieloma więcej! Posiada zaawansowane możliwości agregowania danych i umożliwia ich pełne wykorzystanie w przystępny sposób.

Niektóre z funkcjonalności:

- Nanoszenie danych na mapę, geolokalizacja,
- Wyświetlanie danych na podstawie występowania w przedziale czasowym,
- Wizualizacje w postaci grafów,
- Szukanie anomalii w danych powiązanych z uczeniem maszynowym,

- Generowanie czytelnych raportów.

Jest to potężne narzędzie współpracujące z bardzo zoptymalizowaną pod kątem szybkości bazą danych. Najlepiej sprawdza się, gdy ma do czynienia ze znaczącymi ilościami rekordów do analizy.

Narzędzie jest darmowe i zostało stworzone przez firmę Elastic. Jest częścią tzw. Elastic Stack i w łatwy sposób integruje się z resztą port folio tej firmy. Dostarczana wraz z własnym serwerem Node.js, dzięki czemu żadna dodatkowa konfiguracja nie jest wymagana.

W pracy wykorzystane jako narzędzie do wizualizowania danych z bazy ElasticSearch, w której przechowywane są logi aplikacji internetowej. Jest to domyślny wybór i jest to część tzw. ELK Stack (*z ang. ElasticSearch, LogStash, Kibana*).

4.4.6 Git

Git [24] jest rozproszonym systemem kontroli wersji. Kontroluje wszystkie zmiany, które zostały wprowadzone w plikach i pozwala na późniejszy powrót do starszych wersji. Jest to darmowy program open source, który nadaje się do dużych i małych projektów. Zapewnia wydajność, szybkość i prostotę obsługi.

Jak wspomniano wcześniej, jest to rozproszony system kontroli wersji [25]. Polega on na tym, że każdy klient, który pobiera kod źródłowy, otrzymuje pełny obraz repozytorium. Nie jest to system przechowujący wyłącznie migawkę, wraz z plikami pobiera pełną historię repozytorium. Dzięki temu, w razie awarii serwera, każdy z klientów posiadających projekt, może przywrócić go bez żadnego problemu. Dzięki temu utracenie projektu jest teoretycznie zredukowane do minimum.

Stworzony przez społeczność rozwijającą system operacyjny Linux, z dużym nakładem pracy jego twórcy, Linusa Torvaldsa. Zostali do tego zmuszeni w momencie, kiedy firma tworząca system kontroli wersji, który był używany do przechowywania jądra systemu, Linux został narzędziem płatnym. Podczas tworzenia nowego systemu celami były:

- Prędkość,
- Prostota,
- Wsparcie dla nie liniowego tworzenia oprogramowania (wiele równoległych gałęzi),
- W pełni rozproszony,
- Wsparcie dla dużych projektów.

Od 2005 roku, kiedy został stworzony, Git przeszedł wiele modernizacji i dojrzał na tyle, że przy konsekwentnemu utrzymaniu prostoty używania, zachował wszystkie wartości z powyższej listy.

W projekcie został użyty jako system kontroli wersji, w którym przechowywany był cały kod związany z tworzeniem prototypu. Wybrany został ze względu na pracę na wielu komputerach, a jego rozproszenie umożliwia to w przystępy sposób.

4.4.7 NuGet

NuGet [26] jest platformą, dzięki której programiści mogą tworzyć, udostępniać i wykorzystywać interesujące biblioteki. Taki kod jest pakowany w paczki, które zawierają skompilowany kod najczęściej w postaci DLL i inne elementy, potrzebne jego używania. W prostych słowach paczka NuGet to pojedynczy plik .zip z rozszerzeniem .nupkg i dodatkowym plikiem manifestu, zawierającym takie informacje na temat biblioteki, jak na przykład numer wersji.

Jest to mechanizm stworzony i utrzymywany przez korporację Microsoft. Wspiera platformę .NET. Jest to serwis publiczny, jednak dostępne są również opcję stworzenia swojego własnego repozytorium, jeśli zajdzie taka potrzeba, i to nawet na własnym serwerze. Jest darmowy.

W projekcie użyty do ładowania zewnętrznych paczek. Jest to domyślny system ich dystrybucji na platformę .NET.

4.5. Bazy danych

W tym rozdziale opisane zostaną bazy danych wykorzystane przy budowie prototypu. W projekcie użyto dwóch rodzajów baz: SQL i NoSQL. Autor opisze trzy narzędzia ich funkcjonalności, dystrybucję oraz krótką historię.

4.5.1 SQL Server 2017

SQL Server 2017 [27] jest to relacyjna baza danych, z którą można sie komunikować poprzez język zapytań SQL. Wspierany jest również autorski język Transact SQL lub w skrócie T-SQL.

Została stworzona i jest dalej rozwijana przez korporację Microsoft, której jest głównym produktem, z grupy baz danych. Rozpowszechniana na podstawie dwóch licencji. Deweloperskiej, czyli do użytku przez programistów, podczas tworzenia oprogramowania na ich lokalnych środowiskach, oraz na podstawie licencji komercyjnej, występującej w różnych wariantach i cenach. Kupowana przez firmy, w celu użycia komercyjnego.

Wersja 2017 to lider branży pod względem wydajności i bezpieczeństwa na systemie Linux oraz kontenerach Docker. Oprócz wspierania tych platform, współpracuje z językami programowania takimi jak:

- Java,
- C/C++,
- C#/VB.NET,
- PHP,
- Node.js,
- Python,
- Ruby.

Jest to jedyna komercyjna baza danych z wbudowaną sztuczną inteligencją, dzięki językom R i Python. Oferuje wysoką spójność danych, dzięki możliwości użycia od rozwiązań z prywatnym hostingiem, aż do chmury.

Od momentu wsparcia nie tylko dla systemów Linux, jest to domyślny wybór, jeśli chodzi o relacyjne bazy danych. Możliwość wykorzystania licencji przeznaczonych na systemy on-premise w chmurze.

Wysoka wydajność zapytań, bez konieczności ich ulepszania, jest uzyskiwana poprzez użycie Adaptacyjnego procesowana zapytań (*z ang. Adaptive Query Processing*) i automatycznej korekcji planu (*z ang. Automatic Plan Correction*). Operacje analityczne dostępne są w locie dzięki technologii in-memory.

Najmniej podatności wśród baz danych przez ostatnie siedem lat według NIST. Enkrypcja danych podczas spoczynku z użyciem Always Encrypted i Transparent Data Encryption (TDE). Dynamiczne maskowanie danych (*z ang. Dynamic Data Masking*) ukrywa wrażliwe dane, a kontrola dostępu, pozwalająca na konfigurację z dokładnością do wierszy, dopełnia zadania.

Analiza danych, pozwalająca wysnuć milion prognoz na sekundę, jest skalowalna, wysoko wydajnościowa i zrównoleglona, dzięki wykorzystaniu języków R i Python. Pozwala na użycie zaawansowanych technik uczenia maszynowego z użyciem GPU.

To tylko niektóre z możliwości nowej bazy od Microsoft. W projekcie użyta do trzymania powiązanych ze sobą danych, które posiadają wzajemne relacje. Ze względu na niekomercyjność prototypu i świetne zgranie tej bazy z innymi narzędziami i .NET Framework, jest to wybór domyślny.

4.5.2 RavenDB

RavenDB [29] jest bazą danych NoSQL. W przeciwieństwie do swoich konkurentów wprowadza do swojego działania transakcyjność pod postacią ACID [28]. Jest to skrót od angielskich atomicity, consistency, isolation i durability, co tłumaczy się na: atomowość, spójność, izolację i trwałość.

Atomowość oznacza, że w przypadku wykonywania operacji na bazie danych w ramach konkretnej transakcji, zostaną wykonane wszystkie operacje albo żadna. Spójność występuje w przypadku, kiedy podczas zmiany danych nastąpi jakiś nie spodziewany błąd, to wszystkie dane powrócą do poprzedniego stanu. Izolacja zapobiega modyfikowaniu danych przez transakcje, jeśli poprzednia transakcja dalej nie zatwierdziła swoich zmian. Trwałość polega na zapisaniu wszystkich zmian w taki sposób, że w razie awarii lub restartu systemu, dane są przywrócone do prawidłowego stanu.

Plusami RavenDB są:

- Wydajność,
- Interfejs użytkownika,
- Transakcyjność,
- Wielo-modelowa architektura,
- Wieloplatformowość,

- Wysoka dostępność,
- Łatwe użycie,
- Zaprojektowanie.

Baza jest w stanie wykonać sto tysięcy operacji zapisu na sekundę i milion odczytu. Nie wymaga przy tym sprzętu najwyższej klasy. Interfejs użytkownika, który jest udostępniany w postaci aplikacji internetowej, jest przejrzysty i łatwy w użyciu. Transakcyjność sprawia że otrzymujemy w bazie NoSQL wszystkie najlepsze cechy baz relacyjnych. Przy pracy z zastanym kodem umożliwia stworzenie warstwy abstrakcji nad bazą relacyjną. Działa na środowiskach Windows, macOS, Windows Tablet i raspberry Pi. Wysoka dostępność oznacza, że jest osiągalna prosto z pudełka i konfigurowalna w parę chwil. W ten sposób można uzyskać pełnoprawny klaster bazodanowy. Dzięki zastosowaniu w języku zapytań 85% rozwiązań z SQL, można zostać ekspertem w kilka dni. Dokumentacja i opis wewnętrznego działania pozwalają rozwiązywać samodzielnie wszystkie problemy związane z użytkowaniem bazy. Posiada znajome z baz relacyjnych indeksy, dzięki którym jeszcze bardziej można zwiększyć wydajność zapytań.

Rozwiązanie zostało stworzone i jest rozwijane przez firmę Hibernating Rhinos. Rozpowszechniane jest na podstawie jednej licencji darmowej, która jest ograniczona zasobami sprzętowymi. Licencje komercyjne swoją cenę uzależniają od ilości wykorzystywanych zasobów sprzętowych.

W prototypie użyta ze względu na przechowywanie niespójnych danych w postaci egzaminów, czy wzorów egzaminów. Pliki JSON były domyślnym wyborem. Jest świetnie zintegrowana ze środowiskiem .NET, do którego udostępnia bibliotekę. Pozwala na pisanie zapytań w LINQ, co sprawia że każdy znający go programista nie będzie miał problemów we współpracy z bazą.

4.5.3 ElasticSearch

ElasticSearch [30] to silnik analityczny, który może być używany do rozwiązywania wielu pojawiających się problemów. Jest rozproszony i udostępnia API RESTful. Jest częścią stosu ELK. Jego cechy to:

- Szybkość działania,
- Skalowalność,
- Odporność na błędy,
- Elastyczność.

Dzięki odwróconym indeksom ze skończoną ilością przetworników danych (*z ang. inverted indices with finite state transducers*), udało się osiągnąć dużą wydajność, przy przeszukiwaniu pełno tekstowym i wyszukiwaniu w K wymiarowych B drzewach, które mogą przechowywać wartości numeryczne, jak i geografizne. Skaluje się poprzez klastry. Będzie działać w taki sam sposób zarówno na laptopie, jak i farmie serwerów posiadającej 200 węzłów klastra. Można go wyskalować do miliardów operacji na sekundę, i dalej być w stanie zarządzać danymi bez kłopotu. Wszystkie operacje związane z komunikacją serwerów między sobą, czy błędami sprzętowymi, są rozwiązywane przez klaster, a dane pozostają bezpieczne i dalej dostępne. Pracuje na wszystkich rodzajach danych, m.in. numerycznych, tekstowych, strukturalnych, nie strukturalnych.

Podobnie jak Kibana, jest częścią stosu ELK i został stworzony przez firmę Elastic. Świetnie integruje się z resztą port folio. Jest to narzędzie darmowe i możliwe jest jego dowolne użycie.

W prototypie wykorzystane wraz z Kibana do zrzucania danych działania aplikacji. Jako, że jest ona stworzona w architekturze mikroserwisowej, która może być w stanie wygenerować ogromne ilości danych, jest to idealny wybór. Dobrze integruje się ze środowiskiem .NET przez bibliotekę udostępnioną przez producenta.

4.6. Wirtualizacja i komunikacja

W tym rozdziale autor przedstawi dwa narzędzia użyte w prototypie. Jedno służy do komunikacji, drugie to kontenery Docker, służące do wirtualizacji. Przedstawione zostaną ich cechy, funkcje i zastosowanie w projekcie.

4.6.1 Docker

Docker [31] to platforma służąca do obsługi kontenerów. Kontenery służą do pewnego rodzaju wirtualizacji. Pozwalają na budowanie aplikacji w taki sposób, że będzie działać w taki sam sposób, niezależnie od środowiska. Upraszcza to w dużym stopniu wdrażanie aplikacji na kolejne serwery. Nie trzeba martwić się o konfigurację systemu.

Kontenery to swojego rodzaju opakowanie, zamykana jest w nich aplikacja, wraz ze swoimi zależnościami. Można odnieść wrażenie, że jest to rodzaj wirtualnej maszyny. Daleko im jednak do tego. Podczas, gdy do używania maszyny wirtualnej trzeba skonfigurować cały system, zaczynając od instalacji systemu operacyjnego, kontenery korzystają z bazowych ode lżonych obrazów, tworząc własne zamknięte środowisko.

Mają szereg właściwości, które są nie możliwe lub bardzo trudne do osiągnięcia w klasycznej maszynie wirtualnej:

- Kontenery dzielą zasoby z systemem operacyjnym na którym pracują, dzięki czemu są bardziej wydajne,
- Mogą być włączane i wyłączane w ułamkach sekund,
- Nie mają narzutu, który jest obecny podczas działania bezpośrednio na systemie hosta,
- Możliwość łatwego przenoszenia zmiesza ryzyko błędów wynikające ze zmian w systemie,
- Eliminacja "u mnie działa",
- Dzięki lekkości kontenerów, może ich pracować wiele na jednej maszynie, co oznacza że istnieje możliwość odzwierciedlenia środowiska produkcyjnego na maszynie dewelopera,
- Tworzenie środowiska nigdy nie było tak proste jak pobranie obrazu i wystartowanie kontenera.

Należy pamiętać, że maszyny wirtualne i kontenery zostały stworzone do innych celów. Te pierwsze mają za zadanie w pełni emulować obcy system. Zadaniem tych drugich jest sprawienie, żeby aplikacja była możliwie łatwa do przeniesienia i samowystarczalna.

Warto zaznaczyć, że kontenery jako koncepcja wcale nie są nowością. W systemach UNIX od dawna istnieje komenda chroot, która umożliwia proste wydzielenie systemu plików. Od roku 1998 FreeBSD posiada funkcję wydzielenia takiego systemu plików do procesu. Już od tego czasu wiele firm próbowało swoich sił w stworzeniu podobnego rozwiązania. Dopiero Docker poskładał wszystkie elementy układanki w całość.

W pracy użyty do zamknięcia poszczególnych elementów aplikacji w kontenery. Przy architekturze mikroserwisowej ma to kolosalne znaczenie, ponieważ dzięki temu rozwiązanie można dużo prościej skalować wraz z wzrostem liczby klientów. Jest to najpopularniejsze narzędzie na rynku które posiada 90% rynku.

4.6.2 RabbitMQ

RabbitMQ [32] to serwer do obsługi kolejki komunikatów. Jest implementacją protokołu AMQP 0-9-1 (*z ang. Advanced Message Queuing Protocol*). Definiuje on sposób w jaki komunikaty powinny być kolejkowane, przekazywane, dostarczane. Opisuje również bezpieczeństwo i niezawodność z jaką powinny pracować. RabbitMQ oferuje również wtyczkę do obsługi AMQP w wersji 1.0. Jest to otwarty standard, który umożliwia komunikację pomiędzy innymi serwerami implementującymi go. Definiuje, ile razy komunikat może być wysłany z jednego endpointa do innego:

- Zero lub raz,
- Dokładnie raz lub wiele razy.

Protokół AMQP pozwala klientowi na stworzenie jednokierunkowego połączenia i wysyłania komunikatów do wymiany. Każdy z nich jest osobnym kanałem, który oferuje dodatkowe opcje niezawodności w ich dostarczaniu. Nie ma możliwości wyświetlania list kolejek, wymian czy powiązań. Klient musi dokładnie wiedzieć z czym chce się połączyć.

RabbitMQ oferuje kilka typów wymian:

- Direct exchange, dostarcza wiadomości na podstawie klucza dostarczonego w nagłówku wiadomości,
- Fanout exchange, dostarcza wiadomości do wszystkich kolejek powiązanych z konkretną wymianą,
- Topic exchange, dostarcza wiadomoścido kolejek bazując na filtrowaniu tematów (*z ang. topics*) wymian i kolejek,
- Headers exchange, dostarcza wiadomości bazując na atrybutach nagłówków innych wiadomości.

Ważnym aspektem jest to, że klient może subskrybować kolejk w celu otrzymania komunikatów lub otrzymywać je na żądanie.

RabbitMQ posiada wiele funkcjonalności i wsparcie do pracy na środowiskach produkcyjnych. Niektóre z nich to:

- Wsparcie dla wielu protokołów komunikacji,
- Możliwości routingu,
- Wparcie wielu języków programowania,
- Niezawodność w dostarczaniu komunikatów,
- Klastrowanie,
- Federację,
- Wysoką dostępność,
- Zarządzanie i monitorowanie,
- Autentykacja i kontrola dostępu,
- Rozszerzalność.

Oprócz AMQP, wspierane są protokoły STOMP, MQTT i http, konieczne jest jednak wykorzystanie wtyczek. Wraz z serwerem dostępne są różne biblioteki, dzięki którym można komunikować się z nim z poziomu różnych języków programowania. Posiada możliwość włączenia niezawodności w dostarczaniu komunikatów pomiędzy brokerem, a klientem. Możliwość skalowania poprzez klastrowanie. Federacja, to inna opcja skalowania kolejki, bez konieczności tworzenia klastra. Polega na transferze komunikatów pomiędzy brokerami i kolejkami, w różnych instancjach. Dzięki mirroringowi kolejek, w razie awarii, komunikaty są przesyłane do innych brokerów, w relacji master - slave. RabbitMQ zbudowane jest w architekturze rozszerzalności, dzięki czemu można dokładać funkcjonalności, używając do tego wtyczek.

Jest to rozwiązanie open source, które jest aktywnie rozwijane. W pracy użyte ze względu na swoją obszerną dokumentację i dobrze opisane sposoby implementacji poszczególnych scenariuszy, w różnych językach programowania.

5. Projekt i implementacja

W tym rozdziale zostaną przedstawione poszczególne elementy systemu. Przybliżona zostanie architektura mikroserwisów, jej zastosowanie i narzut, jaki ze sobą niesie. Rozdział kończy się na instrukcji w jaki sposób uruchomić aplikację na środowisku deweloperskim i jak powinny być skonfigurowane narzędzia do jej poprawnego działania.

5.1. Mikroserwisy

Architektura mikroserwisów [33] to metoda wytwarzania oprogramowania, opierająca się na rozdzieleniu komponentów do pojedynczych modułów, skupiających się wokół konkretnej funkcjonalności. Posiadają dobrze zorganizowane interfejsy i operacje. W ostatnich latach ich popularność wzrasta ze względu na firmy, które chcą praktykować kulturę DevOps i metodyki zwinne. Pomagają w budowie aplikacji które są łatwe w skalowaniu, testowalne, a ich aktualizacje można wypuszczać w krótkim odstępie czasowym.

Architektura ta stosowana jest przy aplikacjach o ogromnej skali np. Netflix czy PayPal. Ich systemy przechodziły ewolucję z monolitów do mikroserwisów. Wybór ten jest spowodowany prędkością, z jaką można wypuszczać aktualizacje. Skalowanie odbywa się wokół części systemu, a nie całości. Mikroserwisy powinny być jak najbardziej modularne i umożliwiać wypuszczanie aktualizacji konkretnych części systemu, osobno. Dzięki nim można wybrać najlepsze narzędzie do danego problemu - serwisy nie muszą być napisane w tych samych językach programowania.

Korzyści wynikające z tej architektury to:

- Aktualizacje skupiające się na poszczególnych modułach,
- Łatwe śledzenie kodu ze względu na to, że jedna funkcjonalność jest zamknięta w swoim "pojemniku",
- Możliwość ponownego użycia komponentów,
- Szybsze reagowanie na błędy, wadliwe serwisy można wyizolować,
- Brak zamknięcia (związania) w jednej technologii.

Nie istnieje formalna definicja mikroserwisów, czy standardowy model. Każdy system jest inny. Można jednak zauważyć pewne wspólne cechy:

- Wiele komponentów,
- Skupione wokół biznesu,
- Łatwy routing,
- Decentralizacja,
- Odporność na awarie,
- Rozwojowość.

Jak wspomniano wcześniej, aplikacja zostaje rozbita na komponenty. Dzięki temu można je ulepszać, aktualizować i wypuszczać zupełnie osobno. W ten sposób system nie przestanie być spójny podczas awarii pojedynczej części. Nie ma konieczności wypuszczania całej aplikacji. Podejście to ma jednak swoje wady. Komunikacja musi odbywać sie pomiędzy modułami, a nie

wewnątrz procesu. Zwiększa się również komplikacja systemu przy dystrybuowaniu odpowiedzialności pomiędzy komponentami.

System jest zorganizowany ze względu na potrzeby i priorytety biznesowe. W przeciwieństwie do podejścia monolitycznego, mikroserwisy nie mają podziału na warstwy systemu, jak bazy danych, interfejs użytkownika czy logikę serwerową. Każdy zespół projektowy jest odpowiedzialny za jeden lub więcej komponentów, i opiekuje się nimi przez cały cykl ich życia.

Mikroserwisy mają prostą zasadę działania. Maja otrzymać żądanie, przeprocesować je i odpowiedzieć w poprawny sposób. Jest to przeciwieństwo działania wielu innych architektur, np. opartych o ESB (*z ang. Entreprise Service Bus*), gdzie zaawansowane systemy używają routingu komunikatów i wdrażają warunki biznesowe. Można powiedzieć, ze mikroserwisy posiadają endpointy, które procesują informacje, stosują logikę biznesową i elementy służące do przepływu danych.

Podejście decentralizacji jest faworyzowane przez społeczność skupiającą sie wokół omawianej architektury. Programiści tworzą narzędzia, które pozwalają na eliminację powtarzających się znanych problemów. W przeciwieństwie do monolitu, który korzysta z jednej centralnej bazy danych, w tym podejściu każdy moduł posiada własną bazę danych, którą sam zarządza.

Mikroserwisy muszą być odporne na awarie. Jeśli jeden moduł zawodzi, nie powinno mieć to wpływu na działanie pozostałych. Powinien on się zatrzymać w poprawny sposób i dać sygnał narzędziom monitorującym o końcu swojego życia (lub one powinny to wykryć). W takim przypadku ruch powinien zostać przekierowany do działających kopii. Minusem takiej operacji jest rosnąca komplikacja systemu w porównaniu z monolitem.

Architektura ta ma jednak doskonałe zastosowanie w przypadku systemów, które ciągle się rozwijają, a skala przedsięwzięcia jest ogromna. Nie zawsze można określić na jakich urządzeniach w przyszłości aplikacja ma działać.

Jak każda decyzja w świecie IT, użycie architektury mikroserwisów ma swoje plusy i minusy. Nie jest to rozwiązanie doskonałe, dzięki któremu wszystkie problemy znikają. Zespoły programistyczne muszą być otwarte na komunikację między sobą i skupione na pracy zespołowej. Należy zwrócić szczególną uwagę na wszystkie elementy, które mogą być współdzielone w systemie. Każdy z modułów może mieć różne wymagania dotyczące walidacji danych itp. Dlatego powinno się wydzielić elementy, które będą podlegały wersjonowaniu, co zapobiegnie rozpadowi integralności systemu. Należy zapewnić kompatybilność wsteczną. Niektóre z plusów tej architektury to:

- Programiści są w pełni odpowiedzialni za "swoją" konkretną część systemu,
- Mogą być rozwijane przez stosunkowo mały zespół,
- Moduły mogą być pisane w różnych językach programowania,
- Łatwa integracja i wypuszczanie aplikacji,
- Łatwe do zrozumienia i modyfikowania moduły, nowe osoby szybko stają się produktywne w projekcie,
- Użycie najnowszych technologii,

- Kod zorganizowany wokół możliwości biznesowych,
- Zastosowanie kontenerów w celu szybszego wypuszczania modułów,
- Zmiany wprowadza się na poziomie konkretnej części systemu a nie w całym,
- Izolacja awarii,
- Łatwe skalowanie i integracja z usługami zewnętrznymi,
- Brak długoterminowego poświęcenia względem konkretnego stosu technologicznego,

Niektórymi z minusów przyjęcia tego rozwiązania są:

- Możliwe trudności w testowaniu ze względu na rozproszenie,
- Zwiększająca się ilość modułów może prowadzić do barier informacyjnych,
- Architektura zwiększa złożoność aplikacji przenosząc na programistów odpowiedzialność za elementy związane z zagadnieniami sieciowymi i wydajnościowymi,
- Ze względu na rozproszenie może wystąpić duplikacja nakładów pracy,
- Skomplikowane zarządzanie i integracja systemu jako całości, przy dużej ilości modułów,
- Elementy skomplikowania w monolicie są dodatkowo zwiększone o te występujące w mikroserwisach,
- Zwiększony nakład pracy na elementy związane z komunikacją pomiędzy modułami,
- Implementacja funkcjonalności, która zakłada użytkowanie więcej niż jednego modułu, wymaga koordynacji pomiędzy zespołami,
- Trudna implementacja transakcyjności,
- Zwiększone zapotrzebowanie na pamięć,
- Problem z wspólnymi danymi (synchronizacja).

Podsumowując, architektura ta nie jest idealna. Jak wszystkie rozwiązania, ma swoje zalety i wady. Podrozdział ten miał na celu jedynie przybliżyć temat, ponieważ jest to architektura wykorzystana w prototypie. Bardziej szczegółowy opis mógłby stanowić temat osobnej pracy. Wnioski autora na temat mikroserwisów, zostaną przedstawione w rozdziale 5.4.

5.2. Elementy systemu

W tym podrozdziale autor opisze w jaki sposób zorganizowany jest kod w solucji. Opisane zostaną poszczególne aplikacje wchodzące w skład systemu. Przedstawiony zostanie sposób ich działania, rola w systemie, komunikacja, sposób korzystania z ewentualnej bazy danych i przykładowe przepływy dla kluczowych funkcjonalności.

5.2.1 Organizacja kodu

Kod w solucji podzielony jest na 9 projektów. Pierwszy z nich, `docker-compose`, to konfiguracja, dzięki której możliwe jest uruchamianie w kontenerach Docker. Jest to projekt generowany automatycznie, a najważniejszym plikiem jest `docker-compose.yml`. Można tam używać dowolnych komend stosowanych do zmiany ustawień poszczególnych kontenerów jak np. ustawianie sieci, przekierowanie portów czy stosowanie aliasów, wykorzystywanych przez wewnętrzny DNS Dockera. W ten sposób skonfigurowana jest możliwość startu solucji lokalnie. Po naciśnięciu przycisku start projekty, które wystartują to:

- `Legito.Api`,
- `Legito.BackOffice.Api`,
- `Legito.ProcessorCoordinator.Api`,
- `Legito.AuthorizationServer`,
- `Legito.TextComparer`,
- `Legito.Wiki.TextComparer`.

Kolejny projekt to biblioteka (DLL) `KSolution.Cqrs`. Jest to jeden z projektów, który po skompilowaniu, jest pakowany do paczki NuGet. Nie jest on referowany przez żaden inny projekt. Znajduje się w nim kod, który jest implementowany przez projekty z końcówką Api. Zawiera w sobie implementacje wzorca CQRS wraz z mediatorem. Wszystkie abstrakcje służące do rejestracji poszczególnych komponentów w kontenerze IoC są w jednym miejscu.

Następny projekt również jest biblioteką (DLL) `KSolutions.ServiceDiscovery`. Podobnie jak poprzedni, również po skompilowaniu tworzona jest paczka NuGet i nie jest bezpośrednio referowany z żadnego innego projektu. Paczka natomiast, jest referowana przez projekty Api, które potrzebują komunikacji między sobą. Zawiera abstrakcje służące do komunikacji z aplikacją Consul (odkrywanie usług). W stosunku do obu opisywanych bibliotek decyzja o pakowaniu ich w paczki została podjęta ze względu na konieczność posiadania kompatybilności wstecznej.

Kod we wszystkich trzech projektach API, czyli: `Legito.Api`, `Legito.BackOffice.Api`, `Legito.ProcessorCoordinator.Api` jest zorganizowany w ten sam sposób. Kod był tworzony przez jedną osobę, w związku z czym poszczególne warstwy nie są wydzielone do osobnych bibliotek. Została za to stworzona odpowiadająca tej praktyce struktura folderów. Abstrakcje w postaci interfejsów i klas abstrakcyjnych trzymane są w folderze `Core`. Znajdują się tam również modele, encje i DTO (*z ang. Data Transfer Object*). Jest to warstwa aplikacji, w której znajdują się wszystkie komponenty, które mogą być referowane w wyższych warstwach. Nie znajduje się tam żadna logika. Kolejną warstwą, a raczej folderem ją reprezentującym, jest `Domain`. Znajdują się tam m.in. implementacje interfejsów z warstwy `Core`, konfiguracja schematu bazy danych i obiekt za nią odpowiadający, dostęp do bazy danych poprzez implementacje wzorca CQRS, klasa mapująca obiekty. Krótko mówiąc cały kod, który wykonuje operacje biznesowe. Ostatnią warstwą jest folder `Services`. Znajdują się w niej klasy z nazwami odpowiadającymi nazwom kontrolerów. Służą jako pośrednik do wykonywania operacji pomiędzy kontrolerami a warstwą domenową. Dzięki temu kod w akcjach kontrolerów ograniczony jest do

minimum. Na tym kończy się rozdzielenie na warstwy o którym wspomniano wcześniej. Pozostałymi folderami są:

- `Controllers`,
- `Middleware`,
- `Modules`,
- `Migrations` (jeśli aplikacja korzysta z SQL Server).

W folderze `Controllers` znajdują się kontrolery Api z podziałem na nazwy odpowiadające obsługiwanym przez nie obszarom. `Middleware` to folder w którym znajdują się komponenty rejestrowane przy starcie aplikacji i dołączane do przepływu zapytań HTTP. Przykładem może być warstwa sprawdzająca błędy aplikacji i w razie awarii wysyłająca kod 500, a sam błąd rejestrująca w serwerze logów. Folder `Modules` przechowuje klasy, które służą do rejestracji klas w IoC, z podziałem na warstwy. Ostatnim jest `Migrations`, jak wspomniano powyżej znajduje się on tylko w aplikacjach korzystających z SQL Server, więc nie ma go w `Legito.Api`. Przechowuje dane migracyjne do tworzenia schematu bazy danych wygenerowane przez EntityFramework.

Struktura w procesorach również jest zbliżona, więc zostaną przedstawione razem. Oba są aplikacjami konsolowymi. Podobnie jak w projektach Api została zastosowana architektura z podziałem na trzy warstwy `Core`, `Domain` i `Services`. Zaczynając od najniższej, `Core` również przechowuje abstrakcję i modele, które powinny być dostępne w reszcie aplikacji. `Domain` zawiera wszystkie operacje domenowe, w tym przypadku są to operacje dotyczące skanowania egzaminów. Nie ma tu wzorca CQRS. `Services` przechowuje obiekty służące w komunikacji między klasą startową programu, a operacjami domenowymi czy związanymi z kolejkami. W procesorach znajdują się jeszcze dwa foldery. `Messaging`, który przechowuje obiekty zajmujące się komunikacją z kolejkami i innymi aplikacjami. W tym konkretnym przypadku z `Legito.Processor.Coordinator`. Nie referują one biblioteki odkrywania usług ze względu na jej silne powiązanie z aplikacjami Api. Drugi i ostatni folder to `Modules` przechowujący jak wcześniej klasy służące do rejestracji obiektów w IoC. Każdy z nich posiada swój projekt testowy, w którym są napisane testy integracyjne kluczowych fragmentów kodu.

Autor w każdej aplikacji starał się zachować architekturę trzywarstwową. Poszczególne warstwy nie są wydzielone do osobnych bibliotek, żeby uniknąć niepotrzebnej komplikacji solucji. W projektach znajdują się miejsca z duplikacją kodu. Niestety, zostały zauważone w dalszej części prac i nie zostały poddane re factoringowi. Zostaną one dokładniej opisane w rozdziale siódmym, dotyczącym planów na przyszłość.

Organizacja kodu w aplikacji klienckiej wygląda w zupełnie inny sposób, jako że jest pisana w języku TypeScript. Jest ona podzielona dzięki odpowiedniej strukturze folderów, które grupują w sobie poszczególne funkcjonalności.

5.2.2 Biblioteka KSolution.Cqrs

Jest to projekt, a raczej biblioteka (DLL), która posiada referencję do kontenera IoC, Autofac i SDK Microsoft.NETCore.App. Znajdują się w niej trzy foldery. `Core`, `Domain` i `Modules`. Pierwszy z nich przechowuje wszystkie abstrakcje w postaci interfejsów. `Domain` posiada definicję

tylko jednej klasy i jest nią mediator. W ostatnim folderze jest zadeklarowany moduł, który należy zarejestrować w kontenerze IoC każdej aplikacji, która chce korzystać z biblioteki.

Biblioteka ma za zadanie zamknąć w sobie implementację pochodnej wzorca CQRS połączonego z mediatorem. CQRS polega na rozdzieleniu operacji zapisu, modyfikacji i operacji nie zwracających wyników, od odczytu z bazy danych lub innych usług. Dołączono do tego możliwość tworzenia zdarzeń, które będą wywoływane w zależności od potrzeby w systemie.

Operacje są deklarowane za pomocą komend (*z ang. Command*). Ich implementacja w bibliotece wygląda następująco:

- Interfejs `ICommand` jest implementowany w klasach, które będą później służyły do wywoływania swoich `CommandHandler`,
- Interfejs markujący `ICommandHandler` jest implementowany przez faktyczny interfejs generyczny `ICommandHandler<ICommand>` implementowany w obiektach,
- Interfejs `ICommandHandler` jest implementowany przez klasy, w których będzie wykonywana logika, za pomocą generycznej metody `Handle(T command)`,
- Wspomniany wcześniej moduł za pomocą refleksji rejestruje w kontenerze konkretne komendy wraz z ich handlerami (np. `ICreateExamCommand` z `CreateExamCommandHandler`),
- Interfejs `IMediator` posiada metodę generyczną `Send<TCommand>(TCommand command)`, której implementacja znajduje się w konkretnym obiekcie mediatora,
- Mediator posiada implementację fabryki na podstawie delegatów, dzięki temu w swojej implementacji metody `Send` i kodzie fabryki zawartym w module, jest w stanie odszukać odpowiedni handler i wywołać na nim metodę `Handle`.

Implementacja ta pozwala na bardzo proste dodawanie obiektów zajmujących się modyfikacjami w bazie danych i operacjami. Z perspektywy programisty, dodawanie nowej funkcjonalności wygląda następująco:

- Tworzenie obiektu `Command` z implementacją interfejsu `ICommand` i zadeklarowanie wymaganych pól, np. Id obiektu. Komendy powinny być nie mutowalne,
- Tworzenie `CommandHandler` z implementacją generycznego interfejsu `ICommandHandler`, deklarowanego z komendą stworzoną wcześniej,
- Dodanie logiki do metody `Handle` handlera,
- Logika jest gotowa do wywołania przed mediator przy pomocy metody `Send` i przekazaniu do niej obiektu komendy.

Przykład implementacji znajduje się poniżej. Listing 1 zawiera implementację komendy. Listing 2 to implementacja `CommandHandler`. Listing 3 to przykład wywołania.

Kod 1. Przykład implementacji komendy

```
public class CreateExamProcessingStatusCommand : ICommand
 {
```

```
    public CreateExamProcessingStatusCommand(Guid examId)
    {
        this.ExamId = examId;
    }

    public Guid ExamId { get; private set; }

}
```

Źródło: Opracowanie własne

Kod 2. Przykład implementacji CommandHandler

```
public class CreateExamProcessingStatusCommandHandler :
ICommandHandler<CreateExamProcessingStatusCommand>
    {
        private readonly IProcessorCoordinatorContext _processorCoordinatorContext;

        public CreateExamProcessingStatusCommandHandler(IProcessorCoordinatorContext
processorCoordinatorContext)
        {
            this._processorCoordinatorContext = processorCoordinatorContext;
        }

        public void Handle(CreateExamProcessingStatusCommand command)
        {
            _processorCoordinatorContext.ExamProcessingStatuses.Add(new
     ExamProcessingStatus(command.ExamId));
            _processorCoordinatorContext.SaveChanges();
        }
    }
```

Źródło: Opracowanie własne

Kod 3. Przykład wywołania CommandHandler z poziomu serwisu przy użyciu mediatora

```
public class ProcessingService : IProcessingService
    {
        private readonly IMediator _mediator;

        public ProcessingService(IMediator mediator)
        {
            _mediator = mediator;
        }

        public void ProcessExam(Guid examId)
        {
            var createStatusCommand = new CreateExamProcessingStatusCommand(examId);
            _mediator.Send(createStatusCommand);

            var processCommand = new ProcessExamCommand(examId);
            _mediator.Send(processCommand);
        }

    }
```

Źródło: Opracowanie własne

Przykład pochodzi z projektu `Legito.ProcessorCoordinator.Api` i służy do tworzenia wpisu w bazie danych, odpowiadającemu za status pracy procesorów.

Operacje zwracające wyniki, czyli np. odczyt z bazy danych, są rozwiązywane przez zapytania (*z ang. Queries*). Ich implementacja jest zbliżona do komend. Jednak, w przeciwieństwie do nich, zwracane są wyniki. W bibliotece wygląda to następująco:

- Generyczny interfejs `IQuery<T>` jest implementowany w klasach, które będą służyły do wywoływania określonych `QueryHandler`. Deklarowany w nim jest typ zwracanego obiektu,

- Interfejs markujący `IQueryHandler` jest implementowany przez faktyczny interfejs generyczny `IQueryHandler<TQuery, TResult>` implementowany w klasach `QueryHandler`,
- Interfejs generyczny `IQueryHandler<TQuery, TResult>` jest implementowany przez klasy w których będzie wykonywana logika, za pomocą generycznej metody `TResult Handle(T query)`,
- Za pomocą modułu przy użyciu refleksji rejestrowane są w kontenerze konkretne zapytania wraz z ich handlerami (np. `IGetExamQuery` z `GetExamQueryHandler`),
- Interfejs `IMediator` posiada metodę generyczną `TResult Fetch<TQuery, TResult>(TQuery query) where TQuery : IQuery<TResult>`, której implementacja znajduje się w konkretnym obiekcie mediatora.

Mediator, tak samo jak w przypadku komend, posiada implementację fabryki na podstawie delegatów. Dzięki temu w swojej implementacji metody `Fetch` i kodzie fabryki zawartym w module, jest w stanie odszukać odpowiedni handler i wywołać na nim metodę `Handle`.

Tak jak w przypadku komend, implementacja z perspektywy programisty jest naprawdę łatwa i nie wymaga dużych nakładów pracy. Oto przykład:

- Stworzenie obiektu `Query` z implementacja interfejsu `IQuery`. Zadeklarowanie wymaganych pól np. Id obiektu i typu zwracanego. Zapytania podobnie jak komendy powinny być nie mutowalne,
- Tworzenie `QueryHandler` z implementacją generycznego interfejsu `IQueryHandler` deklarowanego z zapytaniem stworzonym wcześniej,
- Dodanie logiki do metody `Handle` handlera,
- Logika jest gotowa do wywołania przed mediator przy pomocy metody `Fetch` i przekazaniu do niej obiektu zapytania.

Przykład implementacji pochodzący z projektu `Legito.ProcessorCoordinator.Api` znajduje się poniżej. Listing 4 zawiera implementację zapytania. Listing 5 to przykład `QueryHandler`. Listing 6 to wywołanie.

Kod 4. . Przykład implementacji zapytania

```
public class GetProcessingResultsByExamIdQuery : IQuery<IList<ProcessingResult>>
{
    public GetProcessingResultsByExamIdQuery(Guid examId)
    {
        ExamId = examId;
    }

    public Guid ExamId { get; private set; }

}
```

Źródło: Opracowanie własne

Kod 5. Przykład implementacji QueryHandler

```
public class GetProcessingResultsByExamIdQueryHandler :
IQueryHandler<GetProcessingResultsByExamIdQuery, IList<ProcessingResult>>
{
```

```
    private readonly IProcessorCoordinatorContext _processorCoordinatorContext;

    public GetProcessingResultsByExamIdQueryHandler(IProcessorCoordinatorContext
processorCoordinatorContext)
    {
        _processorCoordinatorContext = processorCoordinatorContext;
    }

    public IList<ProcessingResult> Handle(GetProcessingResultsByExamIdQuery query)
    {
        return _processorCoordinatorContext.ProcessingResults
            .Where(x => x.ExamId == query.ExamId).Include(y =>
y.DuplicateAnswers).ToList();
    }
  }
```

Źródło: Opracowanie własne

Kod 6. Wywołanie zapytania przy pomocy mediatora w SendProcessingResultsToBackOfficeCommandHandler

```
var textComparerResults = _mediator.Fetch<GetProcessingResultsByExamIdQuery,
IList<ProcessingResult>>(new GetProcessingResultsByExamIdQuery(command.ExamId));
```

Źródło: Opracowanie własne

Ostatnią funkcjonalnością biblioteki jest możliwość tworzenia zdarzeń. Działają one na podobnej zasadzie jak komendy i zapytania, jednak, w przeciwieństwie do nich, jedno zdarzenie może posiadać wiele handlerów. W bibliotece wygląda to następująco:

- Interfejs `IEvent` jest implementowany w klasach, które będą później służyły do wywoływania swoich `EventHandler`,
- Interfejs markujący `IEventHandler` jest implementowany przez faktyczny interfejs generyczny `IEventHandler<IEvent>` implementowany w obiektach,
- Interfejs `IEventHandler` jest implementowany przez klasy, w których będzie wykonywana logika, za pomocą generycznej metody `Handle(TEvent @event)`,
- Wspomniany wcześniej moduł za pomocą refleksji rejestruje w kontenerze konkretne zdarzenia wraz z ich handlerami (np. `IExamProcessingEvent` z `ExamProcessingEventHandler`),
- Interfejs `IMediator` posiada metodę generyczną `Publish<TEvent>(TEvent @event) where TEvent : IEvent`, której implementacja znajduje się w konkretnym obiekcie mediatora,
- Mediator posiada implementację fabryki na podstawie delegatów, dzięki temu w swojej implementacji metody `Publish` i kodzie fabryki zawartym w module, jest w stanie odszukać odpowiednie handlery i wywołać na nim metodę `Handle`.

Implementacja wygląda następująco:

- Stworzenie obiektu `Event` z implementacja interfejsu `IEvent`. Zadeklarowanie wymaganych pól np. Id obiektu. Zdarzenia, podobnie jak zapytania i komendy, również powinny być nie mutowalne,
- Tworzenie `EventHandler` (może ich być wiele) z implementacją generycznego interfejsu `IEventHandler` deklarowanego z zdarzeniem stworzonym wcześniej,

- Dodanie logiki do metody `Handle` handlera,
- Logika jest gotowa do wywołania przed mediator przy pomocy metody `Publish` i przekazaniu do niej obiektu zdarzenia.

Przykładowa implementacja zdarzenia pochodzi z `Legito.ProcessorCoordinator.Api` i jest zamieszczona na listingach poniżej. Listing 7 to implementacja zdarzenia. Listing 8 i listing 9 zawierają przykłady `EventHandler`. Listing 10 to wywołanie zdarzenia w systemie.

Kod 7: Implementacja zdarzenia

```
    public class SendAnswersToTextProcessorsEvent : IEvent
    {
        public SendAnswersToTextProcessorsEvent(Exam exam)
        {
            this.Exam = exam;
        }

        public Exam Exam { get; private set; }

    }
```

Źródło: Opracowanie własne

Kod 8: Implementacja EventHandler dla powyższego zdarzenia

```
public class SendAnswersToWikipediaTextComparerEventHandler :
IEventHandler<SendAnswersToTextProcessorsEvent>
    {
        private readonly IMessagesQueueManager<Exam> _textMessagesQueueManager;
        private readonly IConfiguration _configuration;

        public SendAnswersToWikipediaTextComparerEventHandler(IMessagesQueueManager<Exam>
textMessagesQueueManager,
            IConfiguration configuration)
        {
            _textMessagesQueueManager = textMessagesQueueManager;
            _configuration = configuration;
        }

        public void Handle(SendAnswersToTextProcessorsEvent @event)
        {

_textMessagesQueueManager.Publish(_configuration.GetSection("MessagingQueueStrings")
.GetSection("WikipediaTextComparer").Value, @event.Exam);
        }
      }
```

Źródło: Opracowanie własne

Kod 9: Implementacja drugiego EventHandler dla powyższego zdarzenia

```
public class SendAnswersToTextComparerEventHandler :
IEventHandler<SendAnswersToTextProcessorsEvent>
    {
        private readonly IMessagesQueueManager<Exam> textMessagesQueueManager;
        private readonly IConfiguration configuration;

        public SendAnswersToTextComparerEventHandler(
            IMessagesQueueManager<Exam> textMessagesQueueManager,
            IConfiguration configuration)
        {
            this.textMessagesQueueManager = textMessagesQueueManager;
            this.configuration = configuration;
        }

        public void Handle(SendAnswersToTextProcessorsEvent @event)
        {
```

```
textMessagesQueueManager.Publish(configuration.GetSection("MessagingQueueStrings").GetSecti
on("TextComparer").Value, @event.Exam);
        }
    }
```

Źródło: Opracowanie własne

Kod 10: Wywołanie zdarzenia w systemie przy użyciu mediatora

```
var textProcessingEvent = new SendAnswersToTextProcessorsEvent(
new Exam(exams.ExamId, exams.Answers.Where(x => x.Type == AnswerType.Text).ToList()));
_mediator.Publish(textProcessingEvent);
```

Źródło: Opracowanie własne

Jak widać, implementacja poszczególnych operacji jest podobna. Wyróżnia je jedynie typ zadań, do jakich mają być używane. Nic nie stoi na przeszkodzie, żeby w komendach, które nie zwracają wyników, używać zapytań do ich otrzymania itd.

Największym plusem tego rozwiązania jest granulacja i możliwość ponownego użycia fragmentów kodu konkretnych komend, zapytań czy zdarzeń. Nie trzeba za każdym razem rejestrować ich w kontenerze IoC - jest to rozwiązane w module zaimplementowanym w bibliotece i zajmuje się tym refleksja.

5.2.3 Biblioteka KSolution.ServiceDiscovery

Podobnie jak biblioteka CQRS, została stworzona do implementacji odkrywania usług w aplikacjach internetowych, które potrzebują się ze sobą komunikować w środowisku mikroserwisów. Tak samo jak ona jest pakowana do paczki NuGet, w celu zachowania kompatybilności wstecznej. Podczas skalowania aplikacji, czyli w tym przypadku dodawania kolejnych kontenerów z aplikacjami w celu ich komunikacji między sobą, potrzebna jest zcentralizowana baza ich adresów.

Odkrywanie usług w prototypie zaimplementowane zostało z wykorzystaniem programu Consul, który został opisany w rozdziale poświęconym narzędziom. Biblioteka posiada referencje do następujących paczek NuGet:

- Consul,
- DnsClient,
- IdentityModel.

Pierwsza biblioteka jest samo tłumacząca - służy do komunikacji z aplikacją. Kolejna służy do uzyskiwania adresu aplikacji. Consul odpytywany jest poprzez Dns. Ostatnia jest dość nietypowa. Każde Api jest chronione za pomocą JWT, które są generowane przez `Legito.AuthorizationServer`. Nie ma wyjątku od tej reguły. IdentityModel wspomaga w komunikacji z biblioteką IdentityServer4, implementowaną przez serwer autoryzacyjny, i służy do tworzenia odpowiedniego zapytania. Dlatego, autor zdecydował się umieścić logikę do ich generowania w tym miejscu.

Biblioteka podzielona jest na dwa foldery. Są nimi `Core` i `Services`. Pierwszy zawiera metody rozszerzające niezbędne, które służą do rejestracji aplikacji podczas jej startu w bazie adresów. Klasę zawierającą wartości stałe. W tym przypadku znajduje się tam zmienna zwracająca adres, pod którym z poziomu kontenera można dostać się do sieci hosta. Enumeratory służące do

oznaczania aplikacji. Modele, które służą do konfigurowania odkrywania usług. Ostatnie są abstrakcje, czyli interfejsy odpowiadające serwisom. Dugi folder `Services` zawiera w sobie implementacje serwisów służących do komunikacji z Consul i generowania JWT.

Dzięki tej bibliotece, implementacja odkrywania usług w aplikacjach, które tego potrzebują, jest niezwykle prosta. Wszystko dzieje się w klasie `Startup.cs` projektu. Konieczne jest jednak dodanie konfiguracji, która jest specyficzna dla każdej aplikacji. Przebiega to następująco:

- W pliku konfiguracyjnym projektu `appsettings.json` należy dodać sekcję `ServiceDiscovery` i zadeklarować w niej pola: `ServiceName`, `HealthCheckTemplate`, tablicę `Endpoints` i Obiekt `Consul`, zawierający pole `HttpEndpoint`,
- W metodzie `ConfigureServices` klasy `Statrup.cs` danego projektu trzeba wywołać metodę rozszerzającą `AddServiceDiscovery` na obiekcie services i przekazać w jej parametrze obiekt, zawierający dane z pliku konfiguracyjnego JSON,
- Na koniec w metodzie `Configure` klasy `Startup.cs` na obiekcie `app`, należy wywołać metodę rozszerzającą `UseConsulServiceRegistration`, która nie przyjmuje żadnych parametrów.

To wszystko. Dzięki temu mechanizmowi, w przypadku konieczności dodania go do rejestru, zmiany są wprowadzane jedynie w dwóch plikach.

Ponieważ cała logika rejestracji jest zamknięta, najważniejszym elementem jest odpowiednia deklaracja konfiguracji. Zaczynając od początku, `ServiceName` odpowiada za nazwę, pod jaką mikroserwis zostanie zarejestrowany w katalogu. Wiele z nich może posiadać taką samą nazwę, dzięki czemu możliwe jest rozłożenie ruchu (*z ang. load balancing*). `HealthCheckTemplate` jest również bardzo ważnym polem. Należy w nim zadeklarować adres, pod którym Consul może sprawdzać stan aplikacji. Jeśli przestanie ona odpowiadać, zostanie skreślona z rejestru. Aplikacja jest odpytywana raz na pięć sekund. Tablica `Endpoints` zawiera adresy, pod którymi można znaleźć aplikację. Powinno się je zadeklarować wraz z portem. Ostatni jest obiekt Consul, zawierający pole `HttpEndpoint`, w którym należy zadeklarować adres, pod jakim jest dostępny program Consul.

Oprócz rejestracji w odkrywaniu usług, biblioteka oferuje również możliwość generowania JWT. Służy do tego serwis `TokenProviderService`. Posiada on w sobie implementację jednej metody - `GetTokenForApi`. Przyjmuje ona dwa parametry, które oba są enumeratorami: oznaczenie api, z którego będziemy chcieli wysłać zapytanie, i oznaczenie api, do którego będziemy wysyłać żądanie.

W kontekście całego systemu, wydzielenie tej logiki do osobnej biblioteki było niezbędne. Uniknięto znaczącej duplikacji kodu, a jej implementacja jest bardzo szybka.

5.2.4. Legito.Api

Jest to aplikacja WebApi. Pracuje w połączeniu z bazą danych NoSQL RavenDB. Nie posiada ona konkretnego schematu - zapisywane są w niej obiekty, które wcześniej poddawane są serializacji do postaci JSON. Znajdują się one w folderze Entity warstwy `Core`, więc można

przyjąć, że to one deklarują schemat. W praktyce jednak można zapisać tam wszystko. Podczas startu rejestruje się w odkrywaniu usług. Ma połączenie z bazą ElasticSearch, w celu logowania pracy i błędów.

Aplikacja zajmuje się operacjami na egzaminach. Przechowuje ich wzory i składuje wypełnione testy. To cała jej odpowiedzialność. Rozwiązane testy są reprezentowane przez klasę `Exam`. Składa się ona z unikalnego Id, Id studenta, który rozwiązał dany egzamin i kolekcji odpowiedzi na pytania. Odpowiedź jest reprezentowana przez klasę `Answer`. Jej właściwości to unikalne Id odpowiedzi, Id pytania, odpowiedź studenta w postaci tekstowej i enumerator, reprezentujący typ pytania. Na podstawie tego enumeratora, w `Legito.ProcessorCoordinator.Api`, podejmowana jest decyzja do jakich procesorów ma zostać wysłane dane pytanie. W tej chwili w systemie istnieje tylko jeden typ pytania - tekstowe. Za przechowywanie wzorów odpowiada klasa `ExamTemplate`. Jej właściwości to:

- `Tag`, odpowiadający za Id w bazie danych,
- `Title`, tytuł egzaminu,
- `StudentIds`, tablica typu `string` przechowująca Id studentów, którzy mogą podejść do egzaminu,
- `Questions`, kolekcja typu `ExamQuestion`,
- `OwnerId`, odpowiada za przechowywanie Id egzaminatora, do którego należy dany wzór,
- `IsActive` to flaga `boolean`, odpowiadająca za oznaczenie czy egzamin jest aktywny,
- `IsCompleted` to również flaga `boolean`, odpowiadająca za oznaczenie zakończonego egzaminu,
- `IsProcessingCompleted` to ostatnia flaga i odpowiada za oznaczenie, czy wyniki procesowania są już dostępne.

Rozwijając powyższe pole `Questions`, składa się ono z: `Tag`, czyli odpowiednik id w bazie danych, `Text`, reprezentujące treść pytania, `Type`, to typ pytania (podział na otwarte i zamknięte) i kolekcji `PossibleAnswers` typu `ClosedQuestionAnswer`, który z kolei zawiera treść odpowiedzi dla pytania zamkniętego i flagę `boolean`, oznaczającą czy odpowiedź jest poprawna.

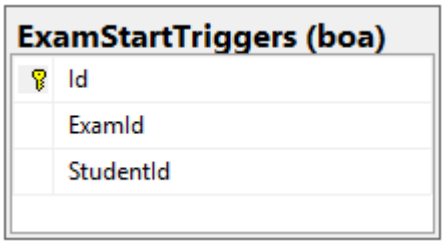
Mikroserwis posiada tylko dwa kontrolery - jeden odpowiadający za operacje na egzaminach, które można rozszerzać, i drugi, sprawdzający czy aplikacja dalej działa (poprzez sprawdzenie stanu zdrowia).

5.2.5. Legito.BackOffice.Api

Jest to aplikacja, której nazwa może być myląca. Całe nazewnictwo poszczególnych Api jest zaszłością po pierwotnej koncepcji i zostało pozostawione celowo. Opis tej decyzji znajduje się w rozdziale trzecim. Mikroserwis jest bramą dostępową do reszty aplikacji z poziomu aplikacji klienckiej. Odpowiada on za kierowanie żądań do odpowiadających im aplikacji. Jest to swojego rodzaju pośrednik.

Podobnie jak poprzednie, api to również podczas startu rejestruje się w odkrywaniu usług i posiada połączenie z bazą ElasticSearch. Serwis posiada dużą odpowiedzialność, przez co może odpytywać wszystkie pozostałe aplikacje, z wyjątkiem procesorów.

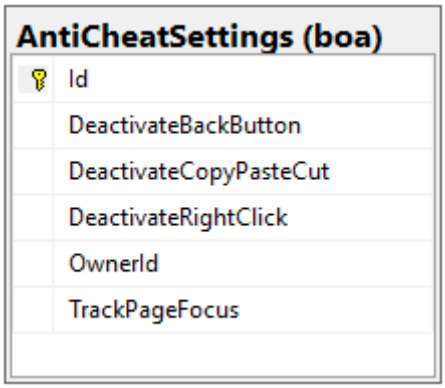
W bazie danych SQL przechowuje informacje o wynikach procesowania, ustawieniach egzaminowania i sprawdza, czy student nie próbuje wypełnić egzaminu po raz drugi. Jej schemat znajduje się na rysunku 2.



Rysunek 2. Tabela ExamStartTriggers

Źródło: Opracowanie własne

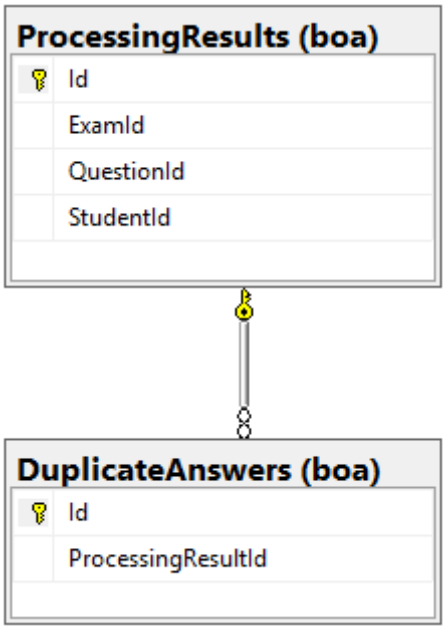
Rysunek 2 przedstawia tabelę `ExamStartTriggers`. Pole Id jest generowanie automatycznie, na poziomie bazy danych, podczas zapisu, `ExamId` to id egzaminu, który student rozpoczął, a `StudentId` to jego id. Tabela jest wykorzystywana podczas rozpoczynania egzaminu, do sprawdzenia, czy student nie wypełnił już egzaminu lub do niego nie podszedł.



Rysunek 3. Tabela AntiCheatSettings

Źródło: Opracowanie własne

Tabela `AntiCheatSettings`, przedstawiona na rysunku 3, służy do przechowywania ustawień egzaminowania. Z jej pomocą, podczas wypełniania egzaminu, włączane są dodatkowe funkcje zapobiegające oszustwom. Pole `Id`, podobnie jak wcześniej, jest generowane automatycznie. Trzy pola `Deactivate` i `TrackPageFocus` służą do włączania poszczególnych funkcji, zapobiegającym oszustwom, które zostaną opisane w dalszej części pracy. `OwnerId` odpowiada za przechowanie id egzaminatora, do którego ustawienia należą.

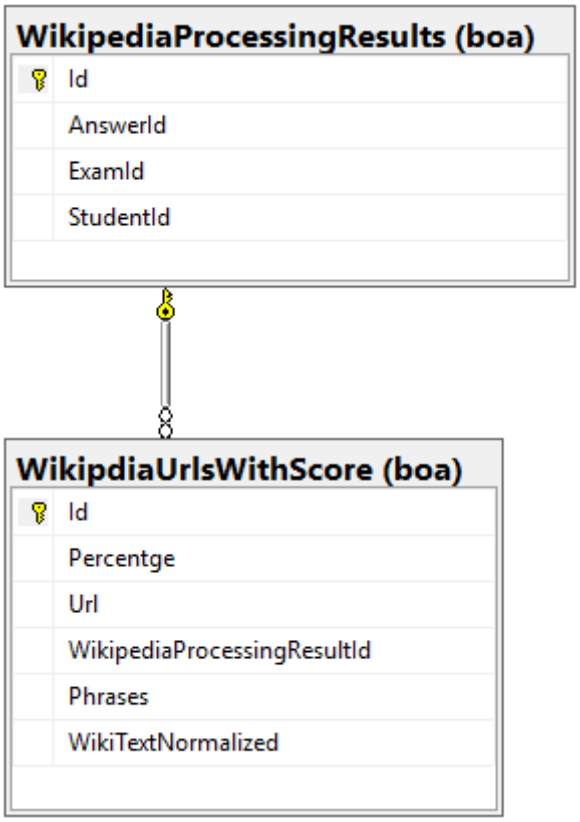


Rysunek 4. Tabele ProcessingResults i DuplicateAnswers

Źródło: Opracowanie własne

Tabele przedstawione na rysunku 4 zostają zaprezentowane razem, ponieważ są silnie ze sobą powiązane i dotyczą jednej funkcjonalności. Mikroserwis przechowuje informacje o wynikach procesowania egzaminów w celu łatwiejszego dostępu. Tabele posiadają relację: jeden do wielu - jeden `ProcessingResults` do wielu `DuplicateAnswers`. Zaczynając od pierwszej od góry tabeli: `Id`, jak poprzednio generowane jest automatycznie, `ExamId` odpowiada id wzoru egzaminu, `QuestionId` to id pytania, `StudentId` to id studenta udzielającego odpowiedzi. Druga tabela zawiera `Id` również generowanie automatycznie i klucz obcy, w postaci `ProcessingResultId` , odnoszący się do tabeli `ProcessingResults`. Opis mechanizmu działania zostanie przedstawiony w dalszej części pracy.

Tabele z rysunku 5 również są przedstawione razem ze względu na ich silne powiązanie, jak w przypadku tabel z rysunku 4. Odpowiadają za przechowywanie wyników z procesora skanującego Wikipedię. Jak we wcześniejszym przypadku, są powiązane relacją jeden do wielu. Jeden wpis w `WikipediaProcessingResults`, może posiadać wiele `WikipediaUrlsWithScore`. Pierwsza tabela posiada automatycznie generowane id. Kolejne kolumny to `AnswerId`, odpowiadający za przechowywanie id odpowiedzi. `ExamId` to id wzoru egzaminu, a `StudentId` to id studenta udzielającego odpowiedzi. Druga tabela przechowuje wyniki skanowania. Id jest generowane automatycznie, `Percentage` odpowiada za celność trafienia, `Url` to link do artykułu w Wikipedii, `WikipediaProcessingResultId` to klucz obcy z wcześniejszej tabeli. `Phrases` to pole przechowujące frazy, pod kątem których było wykonywane skanowanie, a ostatnie pole zawiera znormalizowany tekst odpowiedzi studenta. Mechanizm działania tej funkcjonalności również zostanie opisany w dalszej części pracy.



Rysunek 5. Tabele WikipediaProcessingResults i WikipediaUrlsWithScore

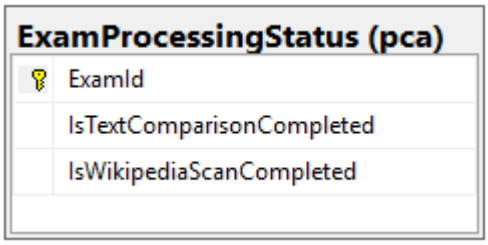
Źródło: Opracowanie własne

Mikroserwis, oprócz roli bramy systemowej, przechowuje też najważniejsze dane dotyczące systemu, umożliwia do nich szybki dostęp i jest odpowiedzialny za konfigurację egzaminowania.

5.2.6. Legito.ProcessorCoordinator.Api

Jak wspomniano wcześniej, mikroserwis ten jest sercem całego systemu i można powiedzieć, że koordynuje jego pracę. Jego odpowiedzialnością jest pobranie na żądanie rozwiązanych egzaminów, pogrupowanie odpowiedzi i wysłanie ich do odpowiednich procesorów. Po skończeniu pracy przez procesory, wyniki są odsyłane i zapisywane w bazie danych. Korzysta z SQL Server i tak jak poprzednie Api, dane błędów i logi wysyła do ElasticSearch.

W swojej bazie danych definiuje pięć tabel, z czego cztery z nich są dokładnie takie same jak w przypadku wcześniejszego `Legito.BackOffice.Api`. Są to pary: `ProcessingResults`, `DuplicateAnswers` przedstawione na rysunku 4. i `WikipediaProcessingResults`, `WikipediaUrlsWithScore` z rysunku 5.

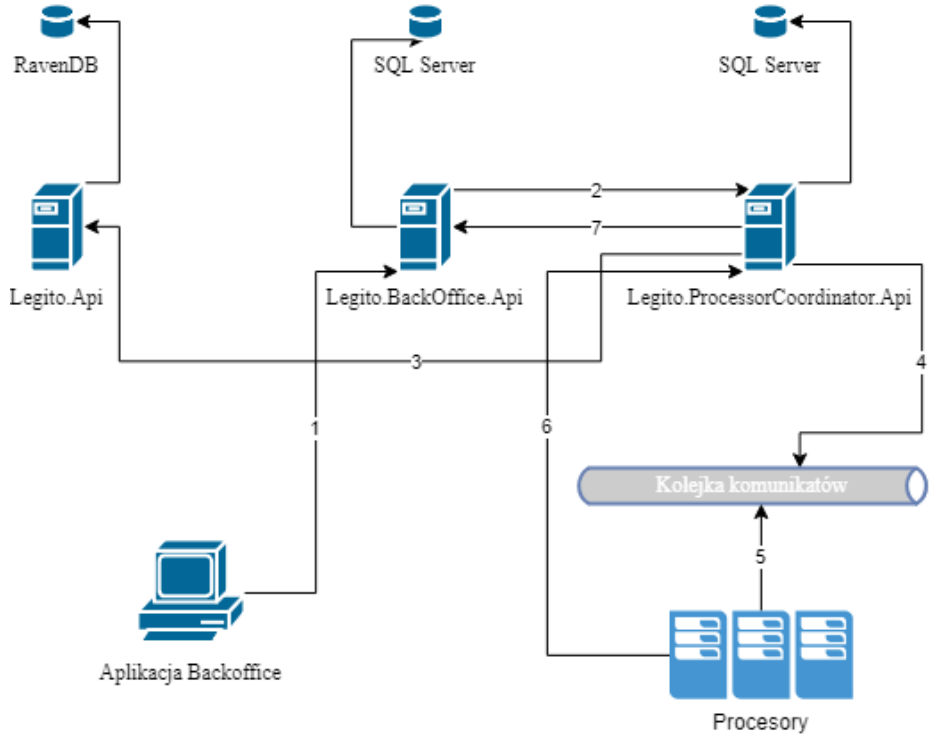


Rysunek 6.Tabela ExamProcessingStatus

Źródło: Opracowanie własne

Tabela przedstawiona na rysunku 6 pełni bardzo ważna rolę. W momencie rozpoczęcia procesowania jest w niej tworzony wpis deklarujący `ExamId`, czyli id egzaminu, a flagi `IsTextComparisonCompleted` i `IsWikipediaScanCompleted` są ustawiane na `false`. Przy zakończeniu pracy przez którykolwiek z procesorów i odebraniu wyników, odpowiednia flaga jest ustawiana na `true`. Jeśli wszystkie posiadają taki status, egzamin jest uważany za przeprocesowany, a wyniki zostają wysłane do `Legito.BackOffice.Api`.

Jedynym zadaniem aplikacji jest koordynacja pracy procesorów. W całym systemie żądanie wygląda w sposób przedstawiony na rysunku 7.



Rysunek 7. Uproszczony przepływ informacji w systemie podczas żądania procesowania egzaminów

Źródło: Opracowanie własne

Na rysunku 7 celowo zostały pominięte elementy systemu, takie jak odkrywanie usług, serwer autoryzacji czy serwer logowania ze względu na to że wprowadziłoby to tylko nie potrzebną komplikację. Cały proces prezentuje się następująco (cyfry odpowiadają strzałkom na rysunku 7):

1. Aplikacja wysyła żądanie procesowania do aplikacji `Legito.BackOffice.Api`,

2. `Legito.BackOffice.Api` przesyła żądanie procesowania do `Legito.ProcessorCoordinator.Api`,
3. `Legito.ProcessorCoordinator.Api` wysyła żądanie do `Legito.Api` z żądaniem wypełnionych egzaminów dla danego id wzoru,
4. Egzaminy są grupowane na podstawie typu pytań i przesyłane na kolejkę w celu procesowania,
5. Procesory zajmujące się danym typem pytania pobierają dane z kolejki i rozpoczynają pracę,
6. Po zakończonym procesowaniu, wyniki w odpowiedniej formie są przesyłane do `Legito.ProcessorCoordinator.Api` i zapisywane w bazie,
7. Po zakończeniu pracy przez wszystkie procesory kompletne wyniki są przesyłane.

Jak widać proces nie jest skomplikowany. Trudniejsza jest część implementacyjna. System ma zapewniać możliwość rozszerzania procesorów i dodawania zupełnie nowych typów pytań, które będą obsługiwane przez inny typ procesora, niż obecnie stworzone. Z poziomu kodu `Legito.ProcessorCoordinator.Api` wygląda to następująco:

- Otrzymywane jest żądanie procesowania, tworzony jest odpowiedni wpis w tabeli `ExamProcessingStatus`,
- Wysłanie żądania egzaminów do `Legito.Api` z id wzoru egzaminu,
- Tworzone jest zdarzenie definiujące wysłanie egzaminów do procesorów,
- Egzaminy są grupowane na podstawie id wzoru egzaminu i typu pytań (otwarte lub zamknięte),
- Wywoływane są konkretne `EventHandler` dla odpowiedniego typu zdarzenia (w systemie występują tylko procesory tekstowe, procesujące pytania otwarte),
- Do kolejki wysyłany jest komunikat z odpowiednio przygotowanymi danymi,
- Wyniki są otrzymywane przez endpoint znajdujący się w kontrolerze `PostProcessingController`,
- W zależności od otrzymanych wyników, wykonywane są mapowania danych i zapis w odpowiednich tabelach, jest to osiągnięte dzięki wzorcowi strategii (konkretny typ procesora wykorzystuje ten sam endpoint),
- Sprawdzanie statusu procesowania egzaminów.

Jeśli egzamin jest zakończony, wyniki wysyłane są do aplikacji `Legito.BackOffice.Api`, w przeciwnym wypadku aplikacja oczekuje na zakończenie pracy wszystkich procesorów.

Dzięki użyciu wzorca strategii udało się zredukować nakład pracy podczas dodawania kolejnych procesorów danego typu. Jeśli zajdzie potrzeba dodania zupełnie nowego typu pytania, architektura powinna zostać odwzorowana.

5.2.7 Legito.AuthorizationServer

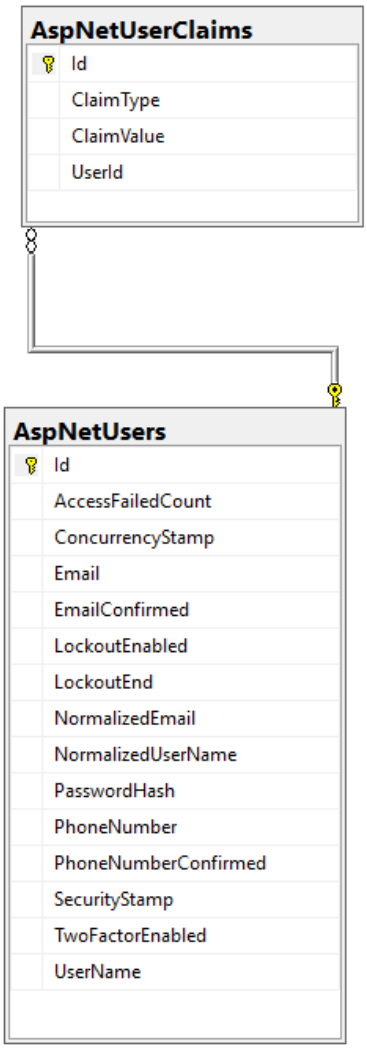
Aplikacja ta jest centralnym punktem przechowywania informacji o użytkownikach. Umożliwia autoryzację aplikacji i autentykację. Korzysta z bazy danych SQL Server. Jego działanie deklaruje biblioteka IdentityServer4, którą implementuje. Wprowadza ona protokoły OpenID Connect i OAuth 2.0. Dzięki niej aplikacja posiada większość funkcjonalności przez nie oferowanych.

W przeciwieństwie do poprzednich aplikacji nie jest to tylko WebApi. Został w niej wykorzystany przykładowy kod dostarczony przez producenta, dostosowany do tego projektu. Jest on napisany z wykorzystaniem wzorca MVC. Posiada dwa obiekty dostępowe do bazy danych. Pierwszy z nich to `ConfigurationDbContext`, który w swoich tabelach przechowuje informacje dotyczące klientów i użytkowników. Schemat bazy danych jest złożony (ze względu na wszechstronność protokołu) i autor pozwoli sobie pominąć jego szczegółowy opis, ze względu na implementację tego rozwiązania, jako tzw. proof of concept. Niemniej jednak, warto wymienić podstawowe tabele, wykorzystywane do autentykacji użytkowników.

Tabele przedstawione na rysunkach 8 i 9, przechowują w sobie najważniejsze informacje, umożliwiające dostęp użytkowników do aplikacji. `AspNetUsers` jest podstawową tabelą zawartą w bibliotece Microsoft.AspNetIdentity, która jest implementowana przez IdentityServer4. Służy do przechowywania użytkowników. W prototypie zostały użyte podstawowe encje oferowane przez bibliotekę. Nic nie stoi na przeszkodzie do zmiany tego podejścia. Dzięki interfejsom tam zawartym możliwa jest pełna konfiguracja schematu bazy danych. Posiada relację do tabeli `AspNetuserClaims`, która odpowiada za przechowywanie roszczeń (*z ang. Claims*), które użytkownik posiada wobec aplikacji. W ten sposób kontrolowany jest dostęp użytkowników do poszczególnych części aplikacji. Tabela `IdentityClaims` służy do przechowywania roszczeń obecnych w systemie. W prototypie są one jednak doklejane bezpośrednio do JWT, co na tym środowisku może być wystarczające, jednak produkcyjnie może być odbierane jako potencjalne zagrożenie.

Rysunek 8.Tabela IdentityClaims

Źródło: Opracowanie własne



Rysunek 9.Tabele AspNetUsers i AspNetUserClaims

Źródło: Opracowanie własne

Mikroserwis w środowisku deweloperskim korzysta z rozwiązania przechowującego informacje o aplikacjach w klasie konfiguracyjnej `IdentityServerConfigurationData`. Zawarte są w niej następujące informacje:

- Informacje o zasobach dotyczących autoryzacji i autentykacji,
- Aplikacje api występujące w systemie,
- Informacje o klientach i ich konfiguracja umożliwiająca uzyskanie JWT.

Proces logowania do aplikacji nie odbywa się na podstawie wysłania do niej informacji i dostaniu w odpowiedzi JWT. Gdy użytkownik wyrazi chęć logowania, zostaje skierowany na serwer logowania, czyli do aplikacji MVC i tam musi wprowadzić informacje logowania. Po potwierdzeniu możliwości dostępu aplikacji klienckiej do roszczeń użytkownika następuje do niej przekierowanie.

Oprócz autoryzacji i autentykacji, aplikacja odpowiada za przekazywanie innym mikroserwisom informacji o użytkownikach. Jest to centralny punkt systemu, odpowiadający za

tego typu zadania. Rejestruje się on w odkrywaniu usług, jednak aplikacja kliencka jako jedyna porozumiewa się z nim bezpośrednio. Na środowisku produkcyjnym skalowanie powinno odbywać się za pomocą rozkładania ruchu za pomocą load balancer.

5.2.8 Procesor odpowiedzi pytań otwartych

W systemie aplikacja występuje pod nazwą `Legito.TextComparer`. Jest to program konsolowy hostowany w kontenerze Docker. Od startu łączy się z odpowiednią kolejką i nasłuchuje nadejścia egzaminu. Schemat działania jest prosty. Egzamin jest pobierany z kolejki, procesowany i na koniec wysyłany za pomocą protokołu HTTP.

Procesor odpowiada za porównywanie odpowiedzi na pytania otwarte studentów odpowiadających na dane pytanie egzaminu. Wykorzystuje do tego algorytm "Strike a Match" [34], stworzony przez Simona White. Dzięki niemu możliwe jest porównywanie części tekstu i ich kolejności. Sprawdza ile par liter występuje obok siebie. Intencją jest rozważenie sąsiadujących liter i wzięcie pod uwagę nie tylko ich rozmieszczenia, ale i kolejności, w jakiej występują w oryginalnym tekście. Przykład przedstawiony przez autora algorytmu dotyczy porównania słów France i French. W pierwszej kolejności litery w słowach są zamieniane na wielkie w celu normalizacji. Drugą operacją jest podzielenie słów na pary liter. Prezentuje się to następująco:

- France - FR, RA, AN, NC, CE,
- French - FR, RE, EN, NC, CH.

Następnie algorytm sprawdza, które pary występują w jednym i drugim tekście. W tym przypadku są to pary FR i NC. Zachodzi konieczność wyrażenia tego w wartości liczbowej. Jeśli funkcja pairs(x) generuje pary występujących obok siebie liter metryką podobieństwa, to jej wzór można wyznaczyć w sposób przedstawiony na rysunku 10.

$$similarity(s1,s2) = \frac{2 \times |pairs(s1) \cap pairs(s2)|}{|pairs(s1)| + |pairs(s2)|}$$

Rysunek 10. Równanie podobieństwa tekstu

Źródło: [33]

Wyniki dla dwóch tekstów s1 i s2 są dwukrotnością części wspólnych, występujących w obu zbiorach, podzielonych przez sumę par liter, występujących w obu tekstach. Dla podanego przykładu podstawienie zmiennych prezentuje rysunek 11.

$$\begin{aligned} similarity(FRANCE, FRENCH) &= \frac{2 \times |\{FR, NC\}|}{|\{FR, RA, AN, NC, CE\}| + |\{FR, RE, EN, NC, CH\}|} \\ &= \frac{2 \times 2}{5+5} \\ &= 0.4 \end{aligned}$$

Rysunek 11. Podstawienie wartości w równaniu

Źródło: [33]

Zakładając że wynik musi mieścić się zawsze pomiędzy 0 a 1, wyrażenie tego w wartościach procentowych jest naturalne. Dla przykładu podanego powyżej wynikiem jest podobieństwo na poziomie 40%.

Oprócz algorytmu porównującego, zaimplementowana jest również koncepcja autora, służąca do grupowania otrzymanych danych i późniejszych wyników, w odpowiedni sposób:

- Otrzymanie danych związanych z egzaminem,
- Pogrupowanie odpowiedzi na podstawie pytania (wynikiem jest kolekcja, w której znajdują sie kolejne kolekcje, zawierające wszystkie odpowiedzi na konkretne pytanie),
- Iteracja po każdej grupie pytań,
- Deklaracja zmiennej `comparisonCheckList`, przechowującej kolekcję obiektów typu `AnswerComparison`, która zawiera dwie właściwości: `SearchAgainst`, czyli odpowiedź, dla której poszukiwane są duplikaty, i kolekcja `DuplicateAnswerIds` zawierająca Id podobnych odpowiedzi,
- Iteracja po każdej odpowiedzi zawartej w grupie pytań,
- Sprawdzenie czy w kolekcji `comparisonCheckList` znajdują się obiekty,
- Jeśli obiekty nie występują, odpowiedź jest do niej dodawana jako obiekt `SearchComparison`, z ustawioną właściwością `SearchAgainst`, wskazującą na tę odpowiedź. Algorytm przechodzi do kolejnej iteracji odpowiedzi,
- Rozpoczyna się iteracja po kolekcji `comparisonChecklist`,
- Jeśli odpowiedź studenta jest podobna do którejkolwiek odpowiedzi z kolekcji sprawdzającej `comparisionCheckList` (do właściwości `SearchAgainst`) w przynajmniej 75%, jej id jest dodawane do kolekcji `DuplicateAnswerIds`, a pętla jest zatrzymywana,
- Jeśli duplikat nie został odnaleziony, odpowiedź jest dodawana do kolekcji jako obiekt `AnswerComparison`, z ustawionym `SearchAgainst`, wskazującym na daną odpowiedź,
- Po zakończeniu iteracji po wszystkich odpowiedziach z kolekcji kontrolnej wszystkie wyniki nie posiadające podobnych odpowiedzi są usuwane,
- Jeśli po usunięciu odpowiedzi z pustymi kolekcjami `DuplicateAnswerIds` zawiera ona jakiekolwiek obiekty, są one dodawane do kolekcji wynikowej,
- Po zakończeniu iteracji po grupach odpowiedzi zwracane są wyniki filtrowane po zawartości kopii.

Wyniki po procesowaniu są konwertowane do odpowiedniego DTO i wysyłane do `Legito.ProcessorCoordinator.Api`, z użyciem odkrywania usług i protokołu HTTP. Wcześniej jednak procesor musi uzyskać JWT od serwera autoryzacyjnego, w celu uzyskania dostępu do aplikacji. W DTO ustawiana jest flaga `boolean IsCheatingFound`. Jeśli ma ona wartość `false`, po otrzymaniu wyników api nie podejmie kolejnych działań związanych z

dostępem bazy, a jedynie ustawi procesowanie jako zakończone. Warto zaznaczyć, że wartość 75%, od której odpowiedzi są traktowane jako duplikat, jest umowna. Wartość ta może być w każdej chwili zmieniona z poziomu kodu, a podmiana na środowisku produkcyjnym nie powinna stanowić problemu, korzystając z narzędzi orkiestrujących jak np. Kubernetes.

5.2.9 Procesor skanujący Wikipedię

W systemie jest to aplikacja pod nazwą `Legito.Wiki.TextComparer`. Jest to program konsolowy hostowany w kontenerze Docker, podobnie jak procesor opisany w poprzednim rozdziale. Podobnie do niego, podczas startu łączy się z odpowiednią kolejką i nasłuchuje nadejścia egzaminu. Schemat działania jest identyczny jak poprzednio. Po pobraniu danych z kolejki, procesor wykonuje swoje operacje, a wyniki wysyłane są za pośrednictwem HTTP.

Aplikacja odpowiada za skanowanie Wikipedii pod kątem fraz zawartych w odpowiedziach na pytania otwarte, udzielonych przez studentów. Wykorzystuje w tym celu Api wystawione przez Wikipedię, a skanowanie odbywa się tylko z wykorzystaniem jej polskiej wersji. Algorytm wykorzystywany do operacji procesowania jest rozwiązaniem autorskim i działa w następujący sposób:

- Otrzymanie danych egzaminu,
- Deklaracja kolekcji przechowującej wyniki procesowania,
- Rozpoczyna się iteracja po kolekcji wszystkich odpowiedzi,
- Jeśli odpowiedz jest pusta lub `null` następuje przejście do kolejnej iteracji,
- Odpowiedz jest dzielona na frazy zawierające 250 znaków, nie obcinając przy tym słów (Api Wikipedii ogranicza przeszukiwanie do 300 znaków),
- Deklaracja kolekcji przechowującej dane wynikowe pojedynczego skanowania,
- Rozpoczyna się iteracja po kolekcji fraz wydzielonych z konkretnego pytania,
- Z wykorzystaniem Api Wikipedii poszukiwana jest pojedyncza fraza,
- Z wykorzystaniem Api zdobywana jest kolekcja numerów stron Wikipedii z odpowiednimi artykułami,
- Rozpoczyna się iteracja po kolekcji numerów stron,
- Na podstawie numeru strony pobierany jest jej tytuł,
- Na podstawie numeru strony i jej tytułu zostaje pobrany tekst artykułu,
- Jeżeli którakolwiek z tych wartości jest pusta, następuje kolejna iteracja po kolekcji numerów stron,
- Otrzymany tekst Wikipedii jest normalizowany,
- Fraza jest dzielona na mniejsze części - zawierające 20 znaków, nie obcinając słów, i normalizowana,
- Ustawienie licznika trafień na zero,

- Rozpoczyna się iteracja po kolekcji frazy rozbitej na mniejsze części,
- Jeśli mniejsza część to jedno słowo, następuje kolejna iteracja,
- W znormalizowanym tekście następuje sprawdzenie występowania części frazy,
- Jeśli się tam znajduje, licznik trafień jest zwiększany o jeden,
- Po zakończeniu sprawdzania fraz, obliczane jest trafienie na podstawie ilorazu liczby trafień przez ilość pomniejszonych fraz,
- Do kolekcji wyników dodawane są wyniki sprawdzenia Wikipedii,
- Po zakończeniu iteracji po pełnych frazach występujących w pytaniu i zasileniu kolekcji wynikowej w pewne obiekty, następuje wyciągnięcie unikalnych adresów URL artykułów do osobnej kolekcji,
- Rozpoczyna się agregacja wyników,
- Tworzona jest nowa kolekcja, przechowująca wyniki skanowania - tym razem docelowa,
- Rozpoczyna się iteracja po kolekcji adresów URL,
- Deklaracja zmiennych pomocniczych `percentages`, `itemCount`, `phrasesAggregated`, `wikiTextNormalized`,
- Rozpoczyna się iteracja po pierwotnej kolekcji wynikowej,
- Jeśli URL elementu pierwotnej kolekcji odpowiada temu z kolekcji unikalnych URL, zmiennym pomocniczym są przypisywane wartości: do `percentages` dodawane jest pole `percentages` z obiektu, `itemCount` zwiększany jest o 1, do `phrasesAgregated` dodawane są frazy z obiektu, a `wikiTextNormalized`, jeśli nie ma przypisanej wartości, przyjmuje tę z obiektu,
- Po zakończonej iteracji pierwotnej kolekcji wynikowej, obliczana jest wartość z agregowanych trafień - jest to iloraz zmiennej `percentages` i `itemCount`,
- Jeśli ich wynik jest mniejszy niż 30%, obiekt nie jest dodawany do kolekcji wynikowej, i następuje kolejna iteracja po kolekcji unikalnych URL,
- Do kolekcji wynikowej dodawany jest nowy obiekt, wykorzystujący dane ze zmiennych,
- Powtórz kroki do końca iteracji po odpowiedziach z egzaminu,
- Zwróć wyniki.

Jak widać algorytm jest dość złożony. Ułatwieniem jego analizy jest jednoczesne patrzenie na kod. Obiekt w kolekcji wynikowej to instancja klasy `WikipediaAnalyzeResult`. Składa się ona z następujących pól: `AnswerId` to unikalne Id odpowiedzi, `StudentId` to unikalne Id studenta i kolekcji `UrlsWithScore` typu `UrlWithHitPercentage`. Obiekt ten jest kluczowy w otrzymaniu wyników skanowania. Zawiera następujące pola: `Url` odpowiadające za przechowywanie adresu artykułu na Wikipedii, `Percentages` wskazujące procent trafienia, `Phrases` to kolekcja fraz po których został wyszukany artykuł, `WikiTextNormalized`

przechowuje z normalizowany tekst pochodzący z artykułu. Wspomniana w algorytmie wartość 30%, która uznaje wynik za trafiony, jest umowna i może być, podobnie jak w przypadku poprzedniego procesora, zmieniona z poziomu kodu.

Normalizacja polega na pozbawieniu tekstu znaków specjalnych. Jako słownik służy kolekcja z zadeklarowanymi znakami: ",", ".", "/", "<", ">", ";", ":", """, "'", "?", "[", "]", "{", "}", "|", "\", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "-", "_", "~", "`", "–". Usuwane są również skróty takie jak: "np", "m.in", "itd", "itp", "j.w". One również są zadeklarowanie w kolekcji jako słownik. Proces normalizacji przebiega w następujących sposób:

- Wartość tekstowa kopiowana jest do lokalnej zmiennej,
- Za pomocą pętli `foreach` zastrzeżone skróty są zamieniane na spacje,
- Za pomocą pętli `foreach` zastrzeżone znaki są usuwane,
- Wszystkie podwójne spacje, które mogą być rezultatem powyższych działań, są zastępowane pojedynczymi,
- Za pomocą wyrażenia regularnego usuwane są wszystkie znaczniki nowych linii.

Wysyłanie wyników procesowania działa dokładnie tak samo jak w przypadku poprzedniego procesora. Jedynym wyjątkiem jest przypisanie wyników do innego pola w DTO. Podobnie jak tam jest również ustawiana wartość `isCheatingFound`.

5.2.10 Odkrywanie usług

Za odkrywanie usług, jak wielokrotnie zostało wspomniane wcześniej, odpowiada aplikacja Consul. System korzysta z dwóch możliwości uzyskania adresu konkretnego mikroserwisu. W przypadku komunikacji aplikacji serwerowych używany jest DNS. Rozwiązanie to zostało wybrane ze względu na to, że ten typ komunikacji korzysta z wbudowanego podziału ruchu (z ang. *load balancing*), oferowanego przez aplikację. Drugim sposobem jest wysłanie żądanie HTTP z prośbą o adres konkretnej aplikacji. W ten sposób komunikuje się aplikacja kliencka. Minusem tego rozwiązanie jest brak podziału ruchu, a jego implementacja musi zostać stworzona w konkretnej aplikacji.

Consul oferuje również sprawdzanie stanu aplikacji poprzez sprawdzenie zdrowia (z ang. *health check*). Jeśli konkretny mikroserwis przestanie odpowiadać, jest skreślany z listy dostępnych aplikacji. Jest to bardzo ważna funkcjonalność, zapewniająca spójność systemu.

W zaawansowanej konfiguracji Consul jest w stanie zastąpić aplikacje lub fizyczne urządzenia odpowiadające za podział ruchu. Autor jednak nie jest w pełni przekonany do tego rozwiązania. Wydaje się, że jest to tzw. wąskie gardło (*z ang. single point of failure*). Pomimo, że jest to aplikacja wysoko dostępna, to występuje jej jedna instancja. Bez replikacji, pojawienie się jakiegokolwiek błędu, oznacza konieczność restartu całego systemu lub rozszerzenie aplikacji poprzez start nowych (rejestrują się w bazie adresów podczas startu).

Jak każde rozwiązanie, ma swoje plusy i minusy, jednak na pewno jest to aplikacja warta sprawdzenia.

5.2.11 Kolejka komunikatów

Kolejką komunikatów w systemie jest aplikacja RabbitMQ. Jest to rozwiązanie kompletne, oferujące ogromną ilość możliwości implementacji różnych scenariuszy biznesowych. W systemie jest wykorzystywane w celu zapewnienia komunikacji z procesorami i możliwości ich skalowania, dzięki wykorzystaniu tzw. work queues.

Założenie jest bardzo proste: N procesorów jednego typu powinno otrzymywać pracę na podstawie dostępności. Jeśli procesor A jest zajęty wykonywaniem pracy, nie otrzyma on zadania X, które zostanie skierowane do procesora B. Każdy typ procesora posiada własny typ kolejki, dzięki czemu są od siebie odseparowane i nie dostaną pracy nie przeznaczonej dla nich.

Wykorzystanie tego podejścia pozwala na łatwe skalowanie procesorów na podstawie zapotrzebowania na ich pracę. Wystarczy uruchamiać kolejne kontenery Docker, zawierające obraz aplikacji. Po uruchomieniu są one od razu gotowe do pracy - zajmuje to kilka sekund.

5.2.12 Serwer logowania

Serwer logowania to baza danych ElasticSearch, a do wyświetlania danych używana jest aplikacja Kibana. Startuje podczas uruchamiania systemu używając kontenerów Docker, a dokładniej `docker-compose`. Jej obraz jest pobierany z Docker Hub, które w dużym uproszczeniu można nazwać odpowiednikiem paczki NuGet dla obrazów Docker.

Do bazy spływają dane logów i błędów z trzech aplikacji w systemie, są to: `Legito.Api`, `Legito.BackOffice.Api` i `Legito.ProcessorCoordinator.Api`. Konfiguracje zdarzeń, które mają zostać tam zapisane, są deklarowane z poziomu każdej z nich osobno.

W celu wykorzystania mechanizmu wystarczy użyć interfejsu `ILogger` z biblioteki Serilog i wykorzystać odpowiedni poziom zdarzenia. Konfiguracja w krokach przedstawia się następująco:

- Deklaracja ustawień w pliku konfiguracyjnym `appsettings.json`,
- Rejestracja interfejsu `ILogger` w metodzie `ConfigureServices` klasy `Startup.cs`.

Deklaracja ustawień w pliku json jest kluczowa, a jej przykład znajduje się na listingu 11.

Kod 11. Konfiguracja Serilog

```
"Serilog": {
  "WriteTo": [
    {
      "Name": "Elasticsearch",
      "Args": {
        "nodeUris": "http://docker.for.win.localhost:9200",
        "indexFormat": "api-{0:yyyy.MM}",
        "typeName": "ApiEvent",
        "batchPostingLimit": 50,
        "period": 2000,
        "inlineFields": true,
        "minimumLogEventLevel": "Warning",
        "bufferFileSizeLimitBytes": 5242880,
```

```
        "bufferLogShippingInterval": 5000
      }
    }
  ]}
```

Źródło: Opracowanie własne

W tablicy `WriteTo` można zadeklarować wiele miejsc w których powinny być przechowywane dane. W przypadku prototypu tablica zawsze posiada jeden obiekt - dotyczący ElasticSearch. To właśnie za to odpowiada pole `Name`. Pole `Args` zawiera informację o URL pod który dostępna jest baza i formacie w jakim mają być zapisywane logi. Reszta ustawień to standard.

5.2.13 Aplikacja kliencka

Aplikacja kliencka znajduje się w osobnym projekcie i repozytorium. Jej robocza nawa to `Legito.Backoffice`. Jest to SPA (*z ang. Single Page Application*), napisana w TypeScript, z wykorzystaniem Frameworka Angular 5.

W systemie stanowi punkt wejścia dla użytkownika. Komunikuje się wyłącznie z `Legito.BackOffice.Api` i `Legito.AuthorizationServer`. W celu wysyłania do nich żądań wykorzystuje odkrywanie usług (z pominięciem autentykacji). Jako, że komunikacja z nim odbywa się poprzez protokół http, konieczne było zaimplementowanie własnego rozdzielenia ruchu. Jest to implementacja naiwna, ponieważ w odpowiedzi na żądanie adresu, otrzymywane są wszystkie adresy, pod jakimi dostępna jest instancja danej aplikacji, a adres jest losowany na podstawie prostej metody losującej. Implementacja algorytmów takich jak round robin w przypadku prototypu nie miała sensu.

Jest to część systemu, służąca tylko i wyłącznie użytkownikom. Na skutek podziału na odpowiednie role w aplikacji, jej interfejs będzie się zmieniał. W przypadku studentów, mają oni dostęp wyłącznie do wyboru i wypełnienia egzaminu. Dla egzaminatorów interfejs staje się dużo bardziej bogaty. Otrzymują oni dostęp do:

- Stworzonych przez siebie egzaminów,
- Kontroli egzaminów (rozpoczęcie, zakończenie, procesowanie),
- Wyświetlania wyników egzaminu,
- Tworzenia egzaminów,
- Ustawień egzaminowania.

Korzystanie i opis tych oraz innych funkcjonalności znajduje się w rozdziale szóstym, który jest poświęcony właśnie tym zagadnieniom.

5.2.14 Dodawanie procesorów

Platforma z założenia miała być rozszerzalna o kolejne elementy, które będą zapobiegać oszustwom w mniej lub bardziej wyrafinowany sposób. Procesory mogą być dodawane na dwa sposoby. Pierwszym z nich jest dodawanie do istniejącej już grupy, czyli w przypadku tego co już jest w systemie, mógłby to być procesor tekstowy dla pytań otwartych. Drugi sposób to dodawanie

zupełnie nowego typu procesora. Jest to zadanie wymagające większego nakładu pracy, jednak wykorzystując wzorce które już są w systemie, wykonalne szybciej niż rozpoczynanie od zera.

Rozpoczynając od podejścia pierwszego, kolejne kroki które trzeba wykonać to:

- Deklaracja encji przechowujących wyniki procesowania w `Legito.ProcessorCoordinator.Api` i `Legito.BackOffice.Api`,
- Rejestracja ich w obiekcie odpowiadającym za komunikację z bazą danych w obu aplikacjach,
- Dodanie odpowiedniej flagi `boolean` do encji `ExamProcessingStatus` w `Legito.ProcessorCoordinator.Api`,
- Dodanie migracji i aktualizacja baz danych,
- Dodanie nowego procesora w enumeracji `ProcessorType`,
- Dodanie klasy realizującej zapis do bazy danych i ustawienie flagi po skończonym procesowaniu, działającej jako część strategii zapisu dla poszczególnych procesorów. Powinna implementować interfejs `IPostProcessingStrategy`,
- Rejestracja wcześniej wspomnianej klasy w module wstrzykiwania zależności `DomainModule`, przy rejestracji słownika trzymającego definicję strategii,
- Deklaracja w konfiguracji aplikacji nazwy kolejki dla nowego procesora,
- Dodanie `EventHandler` odpowiadającego na zdarzenie `SendAnswersToTextProcessorsEvent` i wysyłającego dane na kolejkę,
- Rozszerzenie `PostProcessingDto` o wyniki otrzymywane z nowego procesora,
- Rozszerzenie `SendProcessingResultsToBackOfficeCommandHandler` o wysyłanie wyników nowego procesora do `Legito.BackOffice.Api`,
- Rozszerzenie `StoreProcessedExamResultsCommand` w `Legito.BackOffice.Api` o zapisywanie nowych wyników,
- Dodanie nowego procesora komunikującego się z kolejką i wysyłającego wyniki do `Legito.Processor.Coordinator`.

Proces dodawania nowego procesora do istniejącego typu może wyglądać na skomplikowany, jednak, zaglądając w kod, można dostrzec że cała praca związana z tworzeniem struktury została już zrobiona. Dzięki zastosowanemu wzorcowi strategii odpowiedź na pytanie "od którego procesora otrzymałem te dane" nasuwa się sama. Najtrudniejszym elementem jest dodanie nowego procesora. Jak inne, on również powinien być aplikacją konsolową, a jego praca powinna wyglądać następująco: zabierz dane z kolejki, procesuj, zwróć wyniki. Element procesowania jest kluczowy i najtrudniejszy do opracowania.

Jeśli zachodzi konieczność dodania nowego typu procesora, sytuacja nieco się komplikuje. Do poprzedniej listy dochodzą następujące kroki:

- Deklaracja nowego typu pytania lub operacji w enumeracji `AnswerType` w `Legito.ProcessorCoordinator.Api` i `Legito.Api`,

- Deklaracja nowego zdarzenia, które będzie wywoływało odpowiednie `EventHandler` związane z nim,
- Rozgłoszenie zdarzenia i przekazanie do niego odpowiednich danych egzaminu w `ProcessExamCommandHandler`,
- Możliwa konieczność rozszerzenia elementów odpowiadających za transfer danych.

Powyższe kroki to nieznaczna komplikacja. Dowodzi to tego, że architektura systemu była tworzona z myślą o jego rozszerzalności, i jest to osiągalne stosunkowo nie dużym nakładem pracy. Dodanie tych elementów to chwila pracy dla osoby sprawnie poruszającej się po projekcie. Koniecznie trzeba zaznaczyć ponownie, że najtrudniejszym elementem jest stworzenie procesora samego w sobie, w szczególności zaprojektowanie pracy, jaką ma wykonywać.

5.3. Wystartowanie aplikacji na środowisku deweloperskim

Autor uważa za kluczowe, żeby każda osoba czytająca ten dokument była w stanie nie tylko przeglądać kod aplikacji, ale również wystartować ją na środowisku lokalnym.

Na komputerze należy mieć zainstalowane aplikacje:

- SQL Server 2017,
- RavenDB,
- Docker,
- RabbitMQ,
- Node.js,
- npm,
- Visual Studio 2017.

Z wszystkich wymienionych powyżej to SQL Server wymaga najwięcej uwagi:

- Konfiguracja w celu umożliwiania połączeń tcp/ip,
- Uruchomienie usług Windows powiązanych z tym serwerem - Agent i Browser,
- Utworzenie trzech baz danych dla aplikacji o nazwach: `AuthorizationServer`, `LegitoBackoffice` i `ProcessorCoordinator`.

Wymienione powyżej kroki to podstawa poprawnego działania. Mogą jednak wystąpić niespodziewane komplikacje. Na 3 komputerach, na których baza była tworzona, występowały różne problemy z dostępem do niej. Powyższe kroki powinny zredukować tę możliwość do minimum. Idąc dalej, RavenDB podczas instalacji poprosi o deklarację portu, pod którym ma się znajdować aplikacja. W systemie zadeklarowano port 7777, nic nie stoi na przeszkodzie, żeby go zmienić. Oprócz deklaracji portu, należy stworzyć bazę o nazwie `LegitoApi`. Pozostałe aplikacje korzystają ze swoich domyślnych ustawień.

Przed startem programu należy uruchomić aplikację Consul (dołączona do systemu). Potrzebny będzie do tego plik konfiguracyjny, zawierający informacje o ustawieniach CORS (również dołączony do systemu). Z poziomu wiersza zapytań lub powershell należy nawigować do

folderu, w którym znajduje się Consul.exe i wywołać komendę: `consul agent -dev -config-file "tu podać ścieżkę do pliku json"`. Po uruchomieniu odkrywania usług, można przejść do kolejnego kroku, którym jest uruchomienie solucji Legito i przebudowanie projektu (przy pomocy NuGet ściągnie wszystkie wymagane paczki). Przed budowaniem należy jednak zadeklarować w konfiguracji NuGet Visual Studio, w którym miejscu znajdują się pliki paczek powiązanych z systemem. Należy wskazać folder `LegitoPackages`, znajdujący się w solucji. Wszystkie aplikacje serwerowe są gotowe do startu.

W przypadku aplikacji klienckiej konieczne jest wywołanie trzech komend w wierszu poleceń lub powershell po nawigacji do folderu z projektem. Są to kolejno:

- `npm install -g @angular/cli`,
- `npm update`,
- `ng serve`.

Pierwszą należy wykonać tylko raz - instaluje ona Framework Angular. Druga służy do pobrania wszystkich zależności. Dopiero ostatnia uruchamia aplikację, która będzie działała na porcie 4200. Ważnym aspektem jest wystartowanie aplikacji serwerowych przed kliencką. Podczas startu szuka ona dostępu do serwera autoryzacyjnego. Jeśli go nie znajdzie - nie włączy się.

Po wykonaniu wszystkich opisanych kroków system powinien bez problemu działać na środowisku lokalnym. Oczywiście, można nie używać konfiguracji standardowych, jednak będzie się to wiązało z koniecznością zmian w plikach konfiguracyjnych poszczególnych aplikacji.

5.4. Podsumowanie

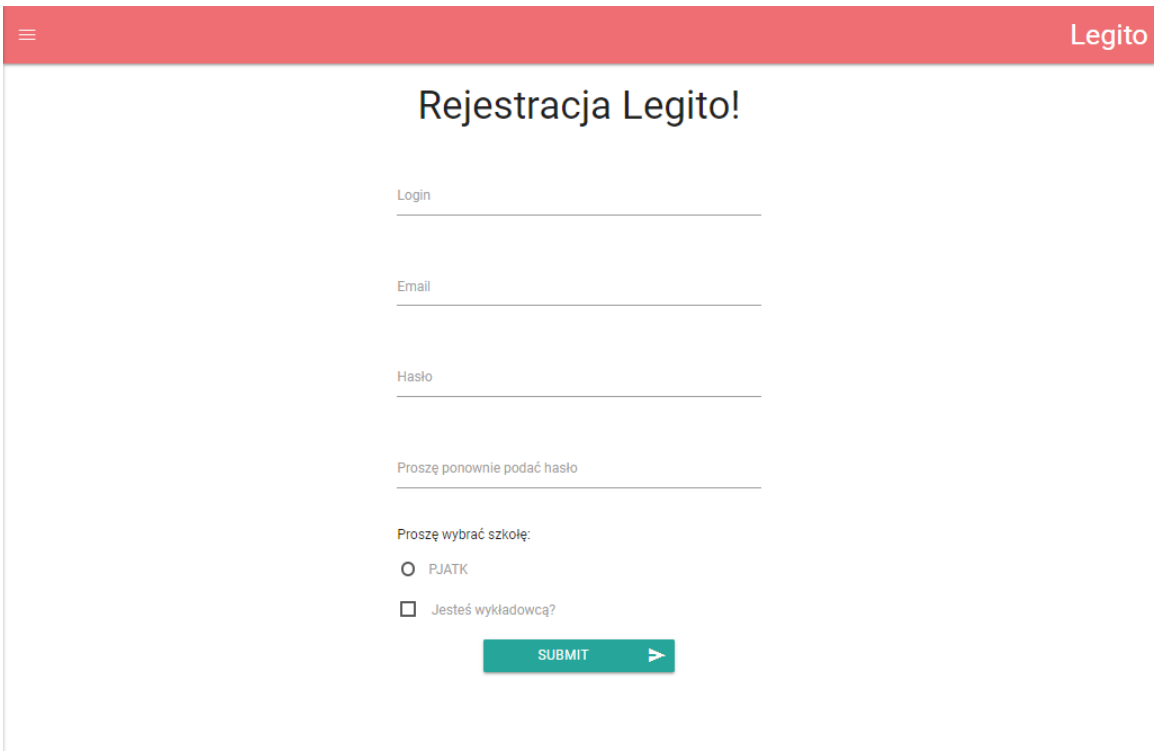
Autor czuje się w obowiązku napisania wniosków na temat użytej architektury, mimo że nie jest to przedmiotem pracy. Był to pierwszy projekt autora tworzony w architekturze mikroserwisów. W przypadku tworzenia prototypu rozwijanego przez jedną osobę, podejście to zupełnie się nie sprawdziło. Spowodowało duże rozciągnięcie prac przez problemy występujące po drodze. Można stwierdzić, że zasymulowana została sytuacja, w której nad systemem pracuje kilka zespołów. W przypadku pracy jednej osoby, autora, nakład pracy był co najmniej kilkukrotny. Jest to na pewno ogromy minus tego podejścia w porównaniu do monolitu. Wniosek jest taki, że architektura ta sprawdza się w momencie, kiedy monolit staje się wąskim gardłem. Należy przez to rozumieć sytuację, kiedy wypuszczanie kolejnych wersji opóźnia się, słabo się skaluje, ceny serwerów są wysokie (chmura Azure czy AWS umożliwia ogromną redukcję kosztów przy podejściu mikroserwisów). Posiadając wiele zespołów, podejście jest jak najbardziej uzasadnione, jeśli te osoby są doświadczone w tej dziedzinie. Według autora, najbardziej przemyślanym podejściem jest przejście z monolitu do mikroserwisów. Odsetek firm, które potrzebują takiego skalowania jakie oferują mikroserwisy, zaraz po wejściu na rynek, jest minimalny i jest nie współmierny do stopnia komplikacji i trudności w monitorowaniu takiego systemu.

6. Funkcjonalności systemu

W tym rozdziale zostaną przedstawione i opisane wszystkie funkcjonalności, znajdujące się w aplikacji klienckiej. Same w sobie nie są skomplikowane, jednak to właśnie tutaj znajdują się kolejne zapory, broniące przed oszustwami podczas wypełniania egzaminu.

6.1. Rejestracja

W celu skorzystania z jakiejkolwiek funkcji aplikacji trzeba być zarejestrowanym użytkownikiem. Klikając w tzw. burger menu, znajdujące się po lewej stronie paska nawigacyjnego, pojawi się wysuwane menu. Poza przyciskiem rejestracji, użytkownicy mają też możliwość zalogowania. Po nawigacji na stronę, pojawi się okno z rysunku 12.



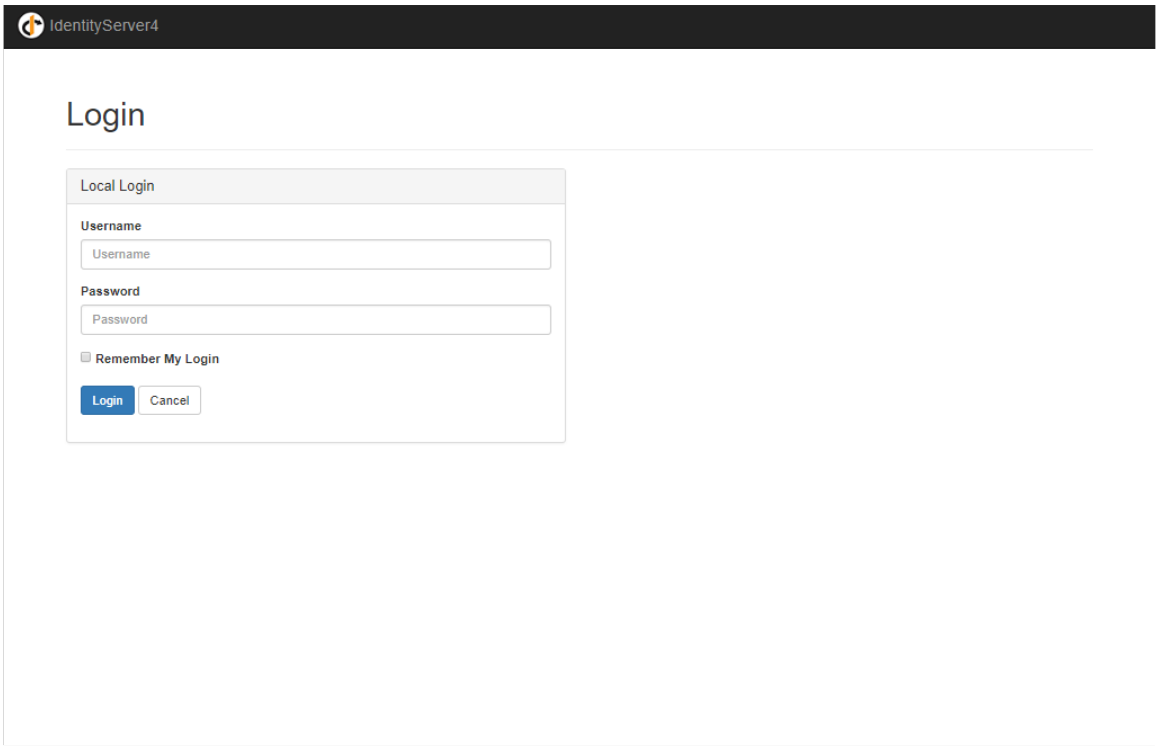
Rysunek 12. Rejestracja w aplikacji klienckiej

Źródło: Opracowanie własne

W kolejności od góry, należy podać login, e-mail, hasło (podwójnie w celu weryfikacji), wybrać szkołę i zadeklarować bycie wykładowcą lub nie. Jak wspomniano wcześniej, wykładowca ma zupełnie inne funkcjonalności w swoim panelu, niż student. Ten wybór ról zapada właśnie podczas rejestracji. Deklaracja szkoły natomiast ma znaczenie podczas dodawania studentów do egzaminu, co zostanie opisane w kolejnych podrozdziałach. Po naciśnięciu przycisku "Submit", na dole formularza, konto zostanie wyświetlone i pojawi się odpowiedni komunikat w górnym prawym rogu ekranu.

6.2. Logowanie

Logowanie jest funkcjonalnością obecną w większości aplikacji. Tutaj, dzięki serwerowi autoryzacyjnemu, odbywa się ono na zasadzie SSO (*z ang. Single Sign On*). Po kliknięciu przycisku "Zaloguj", z menu panelu bocznego, następuje przekierowanie do serwera autoryzacyjnego. Strona przedstawiona jest na rysunku 13.

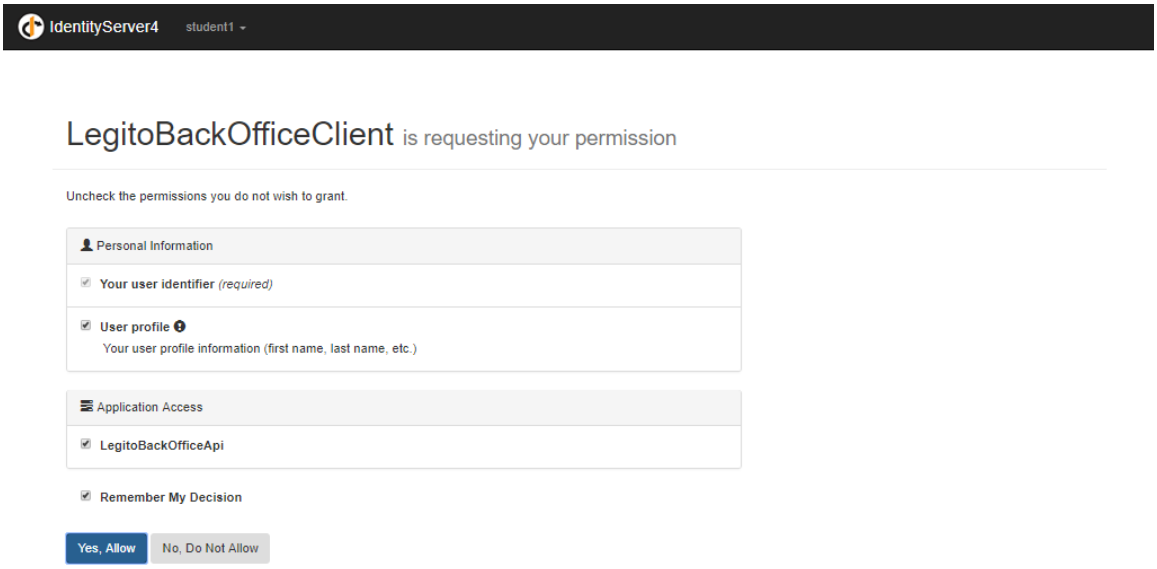


Rysunek 13. Strona logowania

Źródło: Opracowanie własne

Po wprowadzeniu poprawnych danych i kliknięciu przycisku "Login", kolejna strona, która się pojawi, powinna wyglądać jak ta na rysunku 14.

Domyślną opcją jest pozostawienie wszystkich pól zaznaczonych. Deklarowany jest w ten sposób dostęp aplikacji klienckiej do poszczególnych zasobów użytkownika, dostęp do Api i zapamiętanie udzielonych odpowiedzi. Po kliknięciu przycisku "Yes, Allow", następuje przekierowanie do aplikacji klienckiej, a JWT jest zapisywane z URL zwrotnego. Użytkownik jest zalogowany.



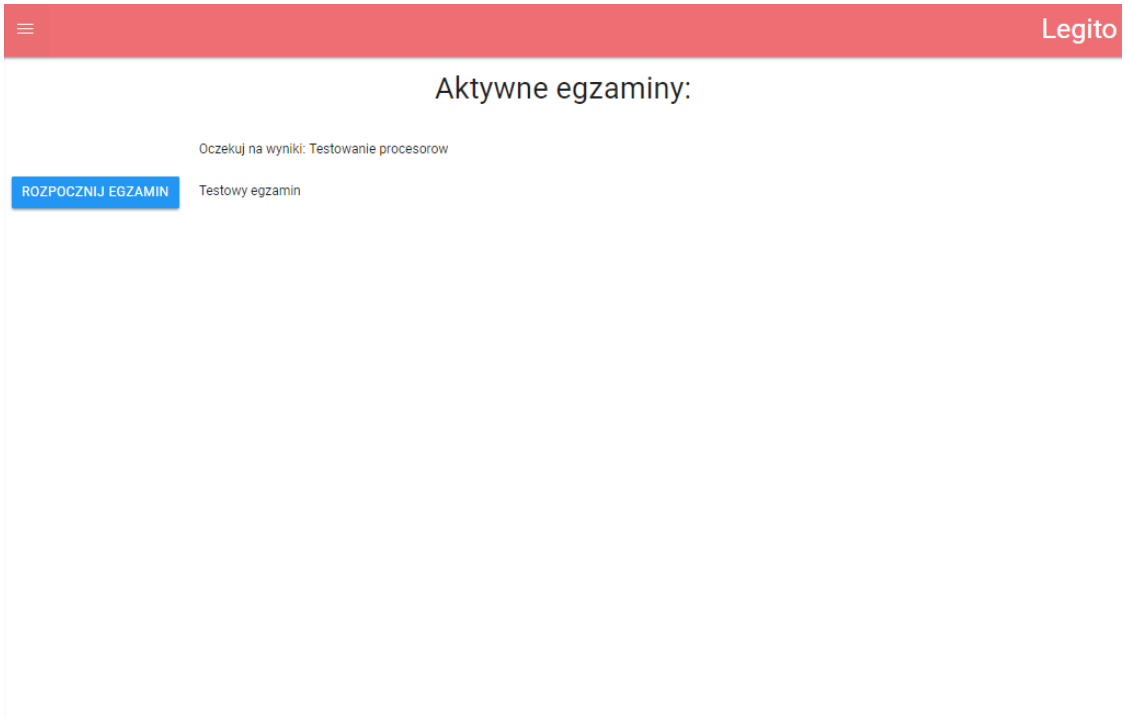
Rysunek 14. Przekazywanie informacji aplikacji klienckiej

Źródło: Opracowanie własne

6.3. Wybór egzaminu

Funkcja ta jest dostępna jedynie dla zalogowanych użytkowników, którzy są studentami. Po zalogowaniu, należy nacisnąć guzik "Aktywne egzaminy", znajdujący się w menu panelu bocznego. Następnie zostanie wyświetlona strona, zawierająca egzaminy, przedstawiona na rysunku 15.

Na liście pojawią się egzaminy. Te, które zostały już wypełnione, będą miały status "Oczekuj na wyniki" przed swoja nazwą, jak na rysunku 15 w pierwszym przypadku. Egzaminy, do których student jeszcze nie podszedł, posiadają po lewej stronie guzik "Rozpocznij egzamin", który spowoduje przeniesienie do części egzaminacyjnej.



Rysunek 15. Aktywne egzaminy

Źródło: Opracowanie własne

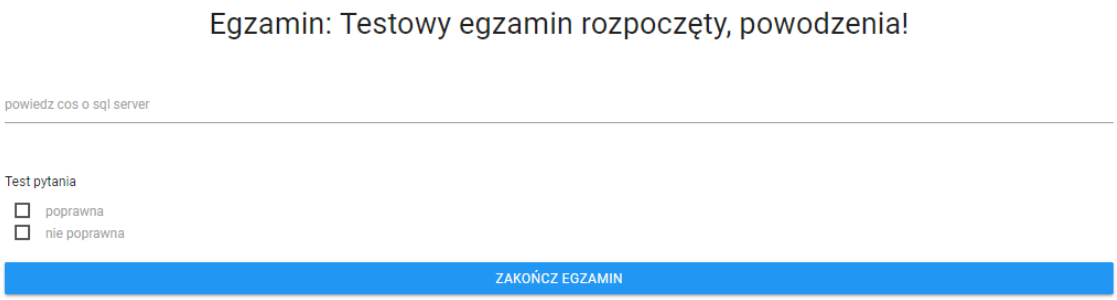
6.4. Wypełnianie egzaminu

Funkcja ta dostępna jest dla zalogowanych użytkowników, posiadających rolę studenta. Po wybraniu egzaminu, jak zostało to przedstawione w poprzednim podrozdziale, następują cztery etapy:

- Sprawdzenie czy student wcześniej nie przystępował do egzaminu,
- Pobranie ustawień egzaminowania,
- Pobranie pytań,
- Wygenerowanie odpowiedniego widoku.

Jak wspomniano na początku, do jednego sprawdzianu nie można podejść dwa razy. Oznacza to, że jeśli student raz naciśnie guzik nawigacji "do egzaminu", nie pojawi się on ponownie. Jeśli będzie próbował nawigować za pomocą URL, również nie uda się wypełnić testu ponownie. Jest to możliwe dzięki zabezpieczeniu sprawdzającemu w bazie danych czy egzamin został już przez studenta wcześniej rozpoczęty.

Po rozpoczęciu egzaminu zostanie wygenerowany jego formularz. Może on wyglądać jak ten przedstawiony na rysunku 16.



Rysunek 16. Egzamin

Źródło: Opracowanie własne

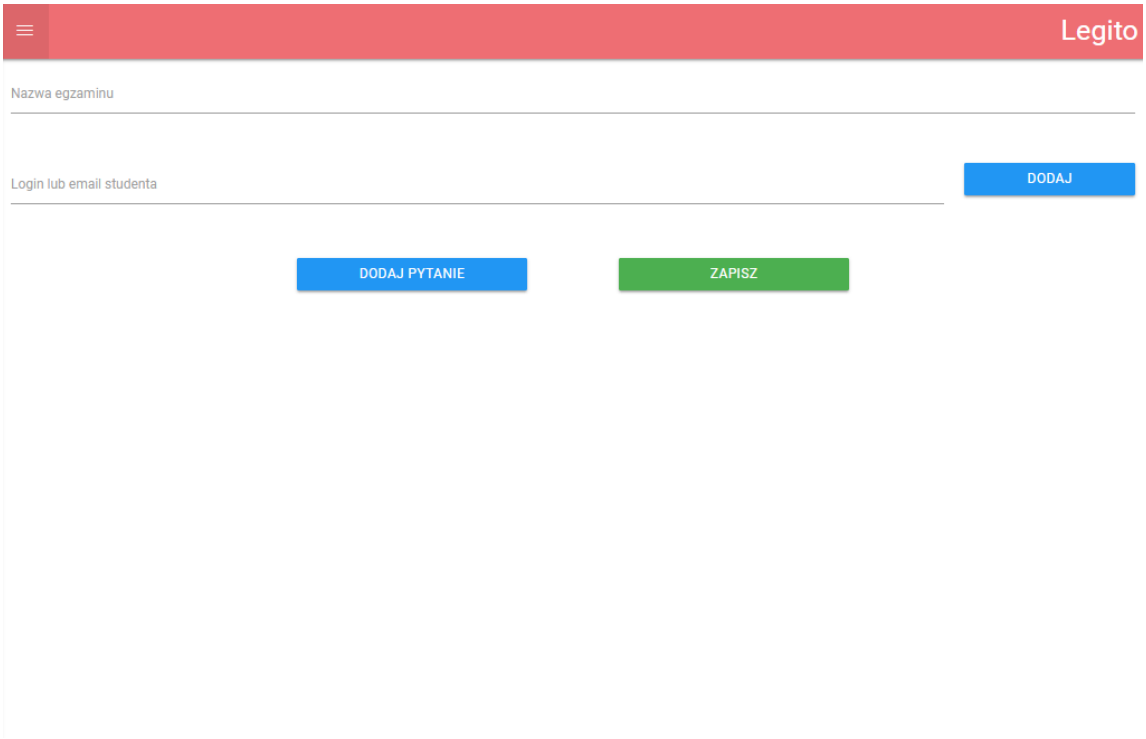
Na stronie przedstawionej na rysunku 16 obowiązują pewne zasady ustalane przez egzaminatora. W zależności od konfiguracji włączone lub wyłączone mogą być opcje:

- Kopiuj, wklej, wytnij,
- Cofnięcie w przeglądarce do poprzedniej strony,
- Prawy przycisk myszy,
- Śledzenie wyjścia ze strony.

Najbardziej interesującą funkcją jest ta ostatnia. Jeśli jest włączona, sprawdza czy egzaminowany wyszedł ze strony. Jeśli zdarzy mu się to raz - w prawym górnym rogu zostanie wyświetlony komunikat ostrzegający przed zakończeniem egzaminu. Jeśli utrata skupienia strony zdarzy sie drugi raz, egzamin jest natychmiastowo kończony i zapisywany w bazie danych. Nie ma możliwości jego ponownego wyświetlenia.

6.5. Tworzenie egzaminu

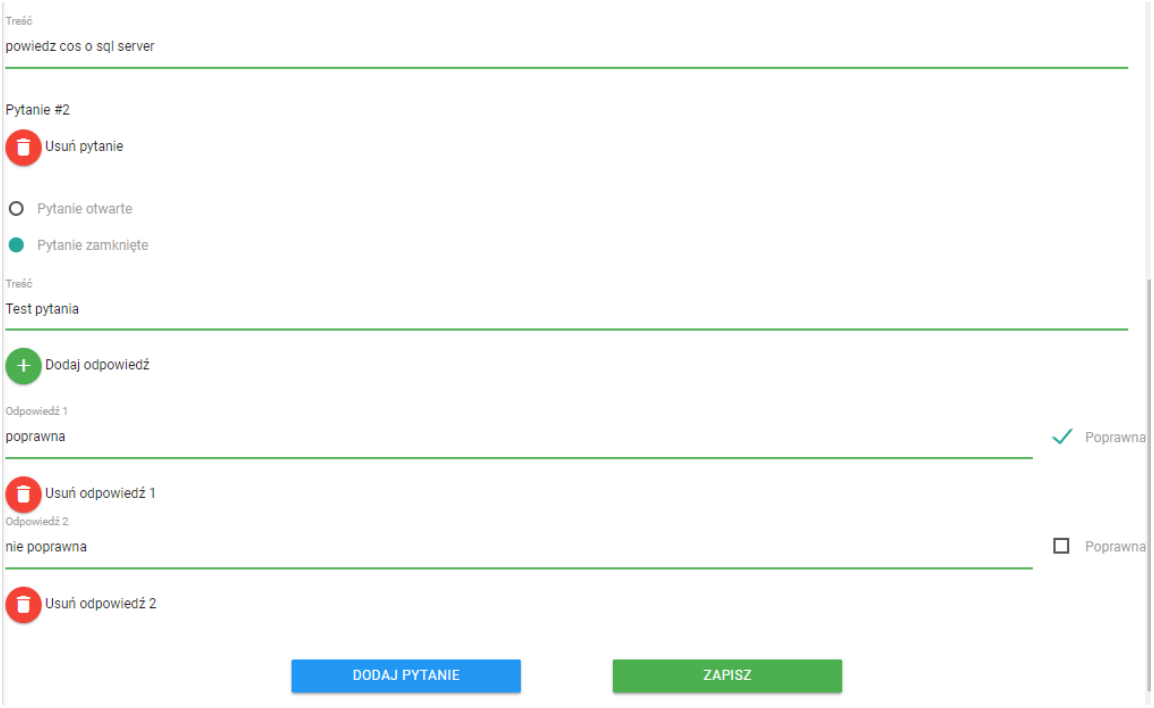
Funkcjonalność dostępna po zalogowaniu dla użytkowników posiadających rolę egzaminatora. Żeby wejść na stronę należy nacisnąć guzik "Tworzenie egzaminów" w menu panelu bocznego. Wygląda ona w sposób przedstawiony na rysunku 17.



Rysunek 17. Tworzenie egzaminu

Źródło: Opracowanie własne

Pierwsze pole dotyczy nazwy egzaminu. Będzie się ono pojawiać zarówno na liście testów egzaminatora, jak i studentów z dostępem do niego. Drugie pole służy do deklaracji kto jest uprawniony do podejścia do egzaminu. Kolejne umożliwia wpisanie loginu lub e-mail użytkownika, w celu dostępu. Po naciśnięciu przycisku "Dodaj" odpowiedni komunikat wyświetli się w prawym górnym rogu. Użytkownik, jeśli jest poprawny, zostanie dodany do listy poniżej tego pola. Przycisk "Dodaj pytanie" po naciśnięciu dodaje formularz, który pozwala na deklarację kolejnego pytania. Posiada przełącznik pomiędzy pytaniem otwartym a zamkniętym. W przypadku tego pierwszego sytuacja jest prosta - pytanie należy wpisać w pole tekstowe. Deklarując typ pytania jako zamknięte, pod jego treścią pojawia się okrągły guzik, umożliwiający dodanie odpowiedzi. Muszą być przynajmniej, dwie z czego jedna powinna być właściwa. Przy każdej odpowiedzi można zadeklarować czy jest ona poprawna, dzięki czemu można tworzyć egzaminy wielokrotnego wyboru. Rysunek 18 przedstawia przykładowe dodane pytania.



Rysunek 18. Dodawanie pytań

Źródło: Opracowanie własne

Aby zapisać dany egzamin, musi on posiadać tytuł, przynajmniej jedno pytanie i przynajmniej jednego studenta, który będzie go wypełniał. Po tej operacji następuje przekierowanie na stronę wyświetlania egzaminów i wyświetlany jest odpowiedni komunikat o zapisaniu operacji.

6.6. Zmiana stanów egzaminu

Egzaminy posiadają cztery stany, które egzaminator może zmieniać z poziomu listy "Moje egzaminy". Umożliwia to znajdujący się pod każdym z nim guzik, zmieniający operację która będzie wykonywał na podstawie statusu egzaminu. Kolejne stany to:

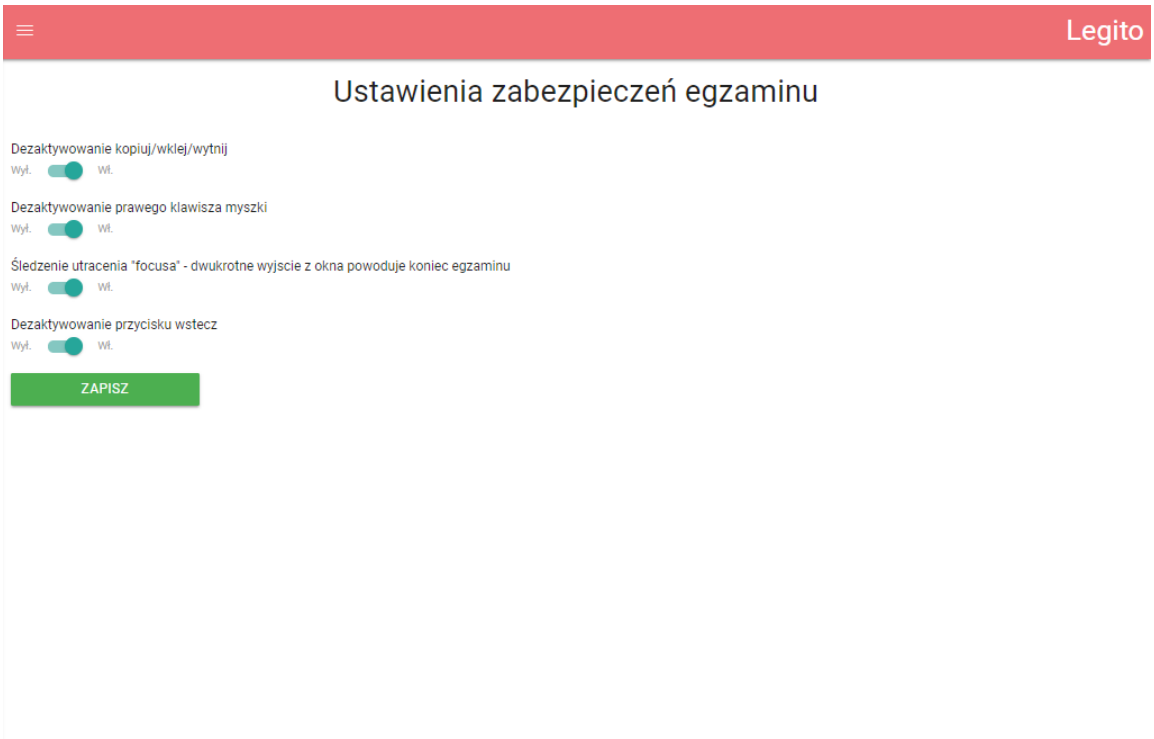
- Rozpoczęcie egzaminu,
- Zakończenie egzaminu,
- Rozpoczęcie procesowania,
- Wyświetlenie wyników.

Odzwierciedlają one również funkcję guzika. Pierwszy odpowiada za umożliwienie studentom dostępu do egzaminu. Drugi wyłącza tę możliwość. Trzeci wysyła do systemu informację o chęci rozpoczęcia procesowania wybranego testu. Po zakończonym procesowaniu, możliwe jest wyświetlenie wyników. Guzik zmienia swój stan po przeładowaniu strony.

6.7. Ustawienia egzaminowania

Każdy egzaminator, po zalogowaniu, ma możliwość zdefiniowania czy poszczególne zabezpieczenia mają być włączone czy wyłączone. Dostęp do nich uzyskuje się poprzez naciśnięcie guzika "Ustawienia egzaminowania", znajdującego się w menu panelu bocznego.

Funkcje te zostały wymienione w podrozdziale 6.4, dotyczącym wypełniania egzaminu. Strona zabezpieczeń wygląda w sposób przedstawiony na rysunku 19.



Rysunek 19. Ustawienia egzaminowania

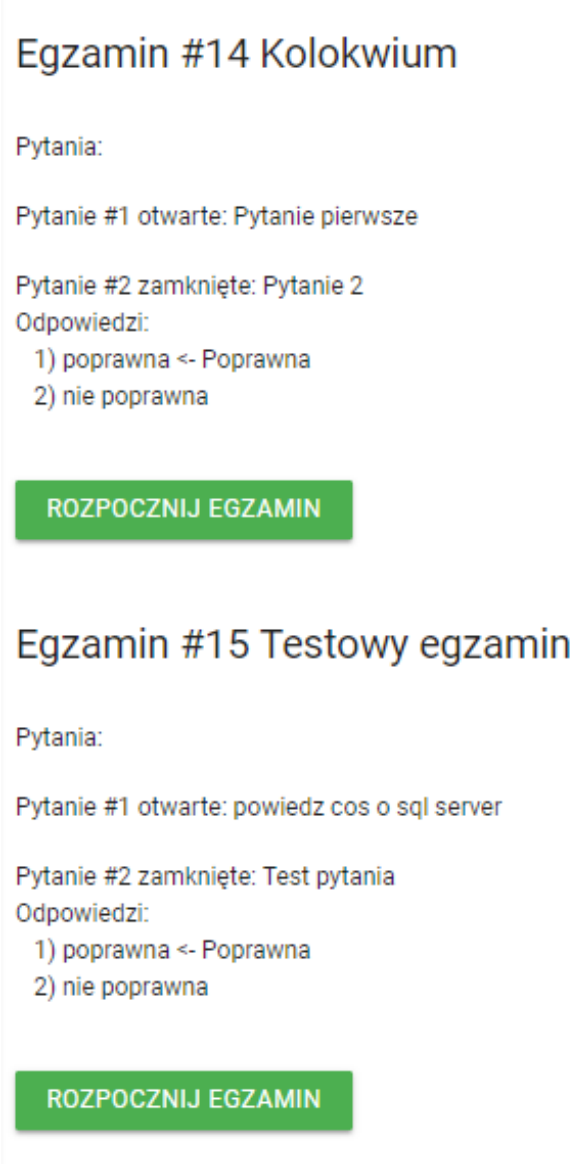
Źródło: Opracowanie własne

Zmianę umożliwiają suwaki, deklarujące włączenie lub wyłączenie poszczególnych opcji. Zapis odbywa się po naciśnięciu guzika "Zapisz", znajdującego sie pod formularzem. Po jego naciśnięciu, w górnym prawym rogu pojawi się odpowiedni komunikat o wykonanej operacji.

6.8. Wyświetlanie wzorów egzaminów

Wyświetlanie wzorów i jednocześnie główne centrum dowodzenia i kontroli nad nimi to funkcjonalność dostępna dla zalogowanych użytkowników z rolą egzaminatora. Przejść do niej można po naciśnięciu guzika "Moje egzaminy", znajdującego się w menu panelu bocznego. Strona wygląda w sposób przedstawiony na rysunku 20.

Dla każdego kolejnego egzaminu wyświetla się jego numer porządkowy wraz z nazwą. Przedstawione są również pytania, jakie się w nim znajdują. Dla odpowiedzi na pytania zamknięte zaznaczone są te poprawne. To na tej stronie odbywa się zmiana stanów egzaminu, opisana w rozdziale 6.6. Możliwa jest też nawigacja do wyników procesowania.



Rysunek 20. Strona moje egzaminy

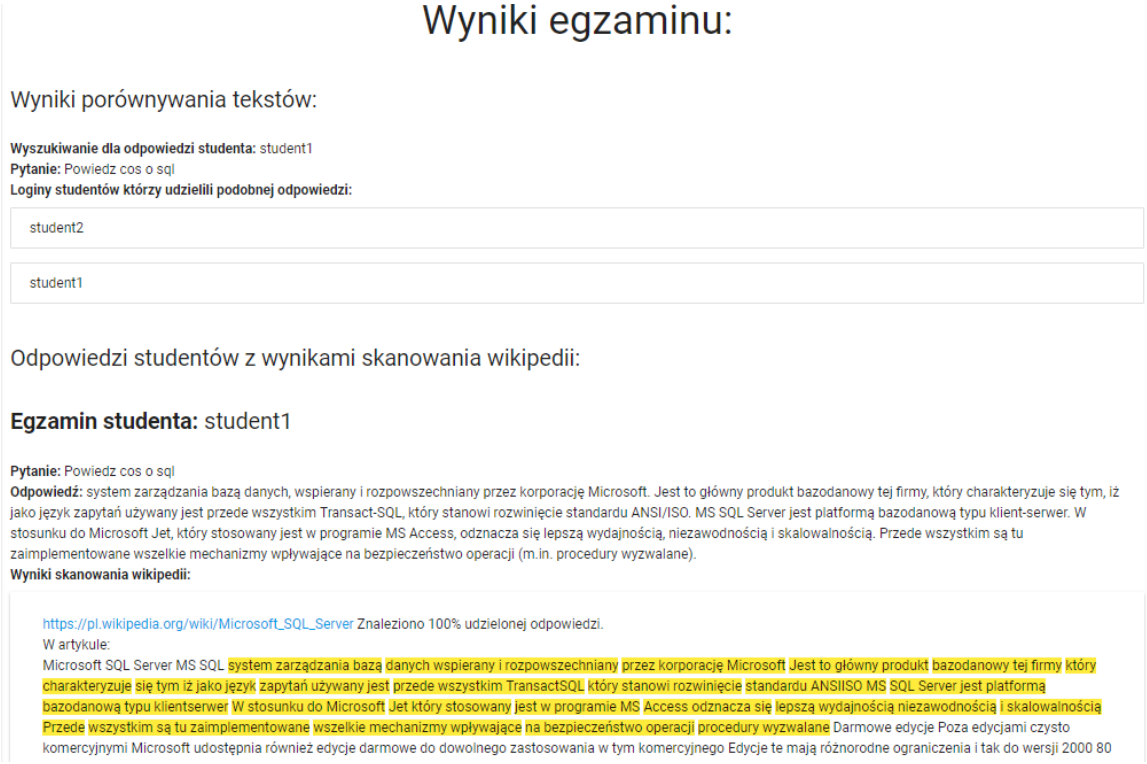
Źródło: Opracowanie własne

6.9. Wyświetlanie wyników procesowania

Funkcja dostępna dla zalogowanych użytkowników posiadających rolę egzaminatora. Umożliwia po skończonym procesowaniu wyświetlenie czytelnych wyników i wskazanie studentów oszukujących. Przejść do niej można po wciśnięciu guzika "Idź do wyników procesowania", znajdującego sie pod wzorem egzaminu. Strona wygląda w sposób przedstawiony na rysunku 21.

Strona posiada dwie główne grupy. Pierwsza z nich wyświetla wyniki porównywana tekstów i znajduje się na samym początku. Wyniki procesora tekstowego wyświetlane są w grupach. Wyszukiwanie dla odpowiedzi studenta deklaruje dla jakiej odpowiedzi prowadzone było przeszukiwanie. Pod nim znajduje się treść pytania i lista loginów studentów, którzy udzielili

odpowiedzi podobnej do studenta, dla którego prowadzone było wyszukiwanie. Grup może być wiele lub żadnej.



Rysunek 21. Wyświetlanie wyników procesowania

Źródło: Opracowanie własne

Druga część to wyświetlenie odpowiedzi studentów na pytania otwarte, opatrzone wynikami skanowania Wikipedii. Dla każdego pytania będzie wygenerowany formularz zawierający: treść pytania, odpowiedź i wyniki skanowania. Skanowanie wyświetlane jest w formie grup wyników. Zawierają one:

- URL do artykułu,
- Trafność,
- Znormalizowany tekst artykułu i pokolorowane odnalezione frazy.

Trafność obliczana jest na podstawie ilości fraz odnalezionych w artykule. Dokładny opis znajduje się w podrozdziale 5.2.9, poświęconemu procesorowi Wikipedii.

Po pytaniach otwartych znajduje się sekcja z odpowiedziami udzielonymi na pytania zamknięte wraz z ich treścią i zostają zsumowane punkty za udzielone odpowiedzi.

7. Plany rozwojowe

W tym rozdziale zostaną opisane plany na przyszłość aplikacji. Autor uważa, że prototyp powinien stać się projektem open source. Przyczyniłoby się to do przyśpieszenia prac przy rozszerzaniu funkcjonalności i podnoszenia jakości. W kolejnych podrozdziałach przedstawione zostaną propozycje, które autor uważa za kluczowe w rozwoju produktu.

7.1. Re faktoryzacja

W programowaniu nic nie jest idealne. Istnieje tyle rozwiązań, ile osób, którym przedstawi się dany problem. Autor wyznaje zasadę: "Już w momencie rozpoczęcia pisania nowego kodu staje się on kodem legacy".

W systemie znajduje się kilka miejsc, w których można podwyższyć jakość kodu. Są to zarówno aplikacja działająca po stronie serwera, jak i aplikacja kliencka. Są to zagadnienia o różnym stopniu trudności, od zmiany flag `boolean`, deklarujących stan egzaminu, po napisanie bibliotek agregujących wspólne abstrakcje działające w procesorach.

Zaczynając od początku, czyli aplikacji serwerowych, w prototypie autor nie widział konieczności implementacji walidacji modeli. Jednak w systemie produkcyjnym jest to niezbędne. Polecane jest użycie do tego biblioteki FluentValidation, która oferuje szeroki wachlarz możliwości i tworzenie skomplikowanych reguł biznesowych.

Kolejna rzecz to sprawdzenie jak pozbycie się wyników skanowania z `Legito.BackOffice.Api` wpłynie na resztę systemu, gdy pozostawione zostaną tylko i wyłącznie w `Legito.ProcessorCoordinator.Api`. Możliwe, że jest to nie potrzebna duplikacja.

Aplikacje podczas komunikacji między sobą mają taki sam schemat działania. Potrzebne jest wygenerowanie JWT, następuje proces serializacji obiektu do postaci JSON (lub nie w przypadku metody GET) i wysłanie żądania za pomocą protokołu HTTP. Logika ta jest na tyle powtarzalna że powinna zostać wydzielona do obiektu, który będzie w sobie łączył kolejne kroki i umożliwiał wysyłanie zapytań HTTP pomiędzy aplikacjami.

Ostatnim pomysłem dla aplikacji serwerowych jest stworzenie biblioteki agregującej abstrakcje procesorów. Obecnie, niektóre fragmenty kodu są bardzo do siebie zbliżone. Wydzielenie logiki komunikacji i przepływów występujących w procesorach jest na pewno możliwe, wymaga to jednak głębszego spojrzenia w kod. Propozycją może być stworzenie interfejsów odpowiadających za komunikację i stworzenie klasy abstrakcyjnej, deklarującej przepływy w aplikacji.

Ponieważ autor czuje się dużo swobodniej w tworzeniu logiki po stronie serwera, niż w aplikacjach klienckich, ta druga wymaga na pewno więcej zaangażowania. Uwaga powinna być skupiona na formularzach, które wyświetlają dane w grupach lub umożliwiają dynamiczne ich dodawanie. Logika obsługująca je znajduje się najczęściej w jednym komponencie i możliwe jest rozbicie jej na mniejsze elementy. Zastosować można więcej mechanizmów, które oferuje Angular.

Ostatnią rzeczą powinno być stworzenie CI/CD. Można wykorzystać do tego orkiestrację, którą oferuje platforma Kubernetes. Autor uważa, że jest to rzecz niezbędna, pracując nad

projektem z innymi programistami. Continous Delivery umożliwi testerom sprawdzanie aplikacji pod kątem błędów, co zredukuje regresję.

7.2. Propozycje procesorów

Procesorem, który początkowo miał znaleźć się w prototypie, jest procesor behawioralny. Jego złożoność i nakład pracy, który musiałby zostać poświęcony jego rozwojowi, wykraczał poza tę pracę.

Idea i zarys są następujące - podczas wypełniania egzaminu zbierane są dane operacji wykonywanych przez egzaminowanego, takie jak: ruchy myszką czy naciskane kombinacje klawiszy. Następnie, na ich podstawie, tworzony jest model zachowań, który świadczy o nieuczciwości. Wymaga to jednak zebrania znaczącej ilości danych i stworzenia poprawnie działającego modelu. Co nie jest zadaniem trywialnym.

Kolejnym procesorem może być crawler, który będzie skanował Internet w poszukiwaniu fraz znajdujących się w odpowiedzi. Czyli bardziej rozbudowany niż ten aktualnie skanujący Wikipedię. Temat jest obszerny i zdecydowanie wykracza poza tę pracę.

Dodatkiem do powyższego crawlera może być procesor, który, w miarę aktualnych możliwości technicznych, tłumaczyłby odpowiedzi na różne języki i przekazywał do wyszukiwania.

Możliwości rozszerzania platformy są nieograniczone. Jedyną blokadę stanowi wyobraźnia i możliwości techniczne.

7.3. Nowe funkcjonalności

W tym podrozdziale zostaną przedstawione propozycje nowych funkcjonalności. Takich, które nie są re factoringiem, ani nie można ich zaliczyć do procesorów. Proponowane są trzy dodatkowe rozwiązania.

Pierwsze z nich dotyczy stworzenia funkcjonalności "wirtualnej sali wykładowej". Polegałaby ona na tym, że podczas wypełniania egzaminu, kamera internetowa w komputerze egzaminowanego przesyłałaby obraz do specjalnie przygotowanej strony dla osób z rolą wykładowcy. W ten sposób egzaminatorzy mogliby w lepszy sposób kontrolować to, co dzieje się podczas egzaminowania.

Druga propozycja jest bardzo zbliżona do pierwszej. Oprócz strumieniowania osób wypełniających egzamin, dostępny jest również ich pulpit. Jest to rozwiązanie idące o krok dalej, jednak w pierwszej fazie mogłoby to być udostępnienie samej przeglądarki z egzaminem.

Ostatnia propozycja to dodanie limitu czasowego dla wypełniania egzaminu. Jest to funkcjonalność, która powinna być zaimplementowana w pierwszej kolejności.

8. Podsumowanie

Zadaniem pracy było zwrócenie uwagi na problem oszustw na platformach e-learningowych podczas wypełniania egzaminów. Istniejące na rynku rozwiązania są produktami komercyjnymi, stosowanymi przez instytucje z całego świata. Ich działanie skupia się wokół nadzorowaniu studenta podczas wypełniania egzaminu. Odpowiadają za to ludzie zatrudniani przez konkretne firmy, stojące za danym produktem, lub zaawansowane algorytmy. Te drugie są w stanie stwierdzić za pomocą nagrania z wypełniania testu czy dana osoba oszukiwała lub podejmowała takie próby.

W pracy przedstawione zostały dwie koncepcje, które różnią się od istniejących rozwiązań podejściem do problemu. Prototyp, który powstał w ramach tej pracy nie skupia się bowiem na używaniu kamery internetowej w celu weryfikacji danej osoby, lecz porównywaniu prac studentów między sobą i konkretnych odpowiedzi z artykułami Wikipedii. Rozwiązanie powinno być nie komercyjne i dystrybuowane na zasadzie open source. W ten sposób, dzięki wsparciu społeczności programistycznej i osób zainteresowanych użytkowaniem systemu, możliwa będzie jego dalsza rozbudowa. Największą zaletą jest rozszerzalność platformy. Umożliwia to dodawanie kolejnych elementów, umożliwiających stwierdzenie nieuczciwości, a ograniczeniami są wyobraźnia i możliwości techniczne. Dzięki wykorzystanej architekturze, kolejne procesory mogą być pisane z wykorzystaniem różnych języków programowania. Nie oferuje tego żaden inny produkt. Ich rozwój zależny jest od firm, które sprawują nad nimi pieczę.

Bibliografia

[1]. Introduction to the C# Language and the .NET Framework. *https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework* Dostęp: 08.07.2018.

[2]. History of C# Programming. *http://aboutcsharpprogramming.blogspot.com/2012/09/history-of-c-programming.html* Dostęp: 08.07.2018.

[3]. Steve Fenton, 2018. Pro TypeScript: Application-Scale JavaScript Development. ISBN-13: 978-1-4842-3248-4.

[4]. Christopher Nance, 2014. TypeScript Essentials. ISBN: 978-1-78398-576-0.

[5]. Ethan Brown, 2016. Learning JavaScript. ISBN: 978-1-491-91491-5.

[6]. SQL - Overview *https://www.tutorialspoint.com/sql/sql-overview.htm* Dostęp: 08.07.2018.

[7]. Introduction to ASP.Net Core *https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1* Dostęp: 08.07.2018

[8]. Pablo Deeleman, 2016. Learning Angular 2. ISBN 978-1-78588-207-4

[9]. Adam Freeman, 2017. Pro Angular. ISBN-13: 978-1-4842-2306-2

[10]. OpenID Connect. *http://openid.net/connect/* Dostęp: 09.07.2018

[11]. OAuth 2.0. *https://oauth.net/2/* Dostęp: 09.07.2018

[12]. IdentityServer4 *http://docs.identityserver.io/en/release/index.html* Dostęp: 09.07.2018

[13]. Entity Framwework Core Quick Overview *https://docs.microsoft.com/en-us/ef/core/* Dostęp: 10.07.2018

[14]. Entity Framwework 6 Quick Overview *https://docs.microsoft.com/en-us/ef/ef6/* Dostęp: 10.07.2018

[15]. Autofac Getting Started *http://autofac.readthedocs.io/en/latest/getting-started/index.html* Dostęp: 11.07.2018

[16]. AutoMapper Getting Started *http://docs.automapper.org/en/stable/Getting-started.html* Dostęp: 11.07.2018

[17]. AutoMapper *https://github.com/AutoMapper/AutoMapper* Dostęp: 11.07.2018

[18]. Serilog *https://github.com/serilog/serilog* Dostęp: 16.07.2018

[19]. Co nowego w programie Visual Studio 2017 *https://visualstudio.microsoft.com/pl/vs/whatsnew/* Dostęp: 17.07.2018

[20]. ReSharper *https://marketplace.visualstudio.com/items?itemName=JetBrains.ReSharper* Dostęp: 17.07.2018

[21]. WebStorm *https://www.jetbrains.com/webstorm/* Dostęp: 17.07.2018

[22]. What is Consul? *https://www.consul.io/intro/index.html* Dostęp: 18.07.2018

[23]. Kibana *https://www.elastic.co/products/kibana* Dostęp: 18.07.2018

[24]. A Short History of Git *https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git* Dostęp: 23.07.2018

[25]. About Version Control *https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control* Dostęp: 23.07.2018

[26]. An introduction to NuGet *https://docs.microsoft.com/en-us/nuget/what-is-nuget* Dostęp: 23.07.2018

[27]. Opis programu SQL Server 2017 *https://download.microsoft.com/download/F/9/A/F9A1B5AA-D57C-4B4D-9C3E-715B800B0419/SQL_Server_2017_Datasheet.pdf* Dostęp: 23.07.2018

[28]. ACID *https://searchsqlserver.techtarget.com/definition/ACID* Dostęp: 25.07.2018

[29]. RavenDB *https://ravendb.net/features* Dostęp: 25.07.2018

[30]. ElasticSearch *https://www.elastic.co/products/elasticsearch* Dostęp: 25.07.2018

[31]. Adrian Mouat, 2016. Using Docker. ISBN: 978-1-491-91576-9.

[32]. Martin Toshev, 2016. Learning RabbitMQ. ISBN: 978-1-78398-456-5

[33]. What are Microservices *https://smartbear.com/learn/api-design/what-are-microservices/* Dostęp: 08.04.2018

[34]. How to Strike a Match *http://www.catalysoft.com/articles/strikeamatch.html* Dostęp: 09.08.2018

[35]. Proctortrack *https://www.proctortrack.com/* Dostęp: 20.08.2018

[36]. Proctoru *https://www.proctoru.com/* Dostęp: 21.08.2018

[37]. AIProctor *https://www.aiproctor.com/* Dostęp: 21.08.2018