



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
INSTYTUT INFORMATYKI

PROJEKT DYPLOMOWY

*System wspomagania organizacji zajęć online oparty na wzorcu
model driven architecture*

Model driven architecture based system for organizing online classes

Autor: Mikołaj Bul, Konrad Czerepak, Adam Niemiec, Szymon Stępień
Kierunek: Informatyka
Typ studiów: Stacjonarne
Opiekun pracy: dr inż. Sławomir Zieliński

Kraków, 2023

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godność studenta.”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelnia przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworem stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....
(czytelny podpis studenta)

Spis treści

Słownik	5
Wstęp	6
1 Cel prac i wizja produktu	7
1.1 Charakterystyka problemu	7
1.2 Istniejące rozwiązania	8
1.2.1 D2L Brightspace	8
1.2.2 Schoology	11
1.2.3 Podsumowanie funkcjonalności	12
1.3 Motywacja projektu	13
1.4 Wizja	13
1.5 Prototyp	15
1.6 Analiza zagrożeń	16
1.7 Podsumowanie rozdziału	17
2 Zakres funkcjonalności	18
2.1 Przedstawienie aktorów	18
2.2 User Stories	19
2.3 Grupy funkcjonalne	21
2.3.1 Podsystem zarządzania modułami:	21
2.3.2 Podsystem zarządzania kursami i zajęciami:	22
2.3.3 Podsystem administracji:	22
2.4 Specyfikacja wymagań niefunkcjonalnych	23
2.5 Współpracujące podsystemy	24
2.6 Podsumowanie rozdziału	25
3 Wybrane aspekty realizacji	26
3.1 Struktura systemu	26
3.1.1 Serwer	27
3.1.2 Orkiestrator	28
3.1.3 Silnik MDA	28
3.1.4 Serwis Chmurowy	28
3.1.5 Baza Danych	28
3.1.6 Aplikacja Webowa	28
3.2 Stos technologiczny	29
3.2.1 Aplikacja Webowa	29
3.2.2 Back-end	31
3.2.3 Serwis Chmurowy	34
3.3 Model danych	37
3.3.1 Schemat bazy danych <i>Serwera</i>	37
3.3.2 Schemat bazy danych <i>Orkiestratora</i>	39
3.4 Implementacja wzorca MDA	40
3.4.1 Modele danych	40
3.4.2 Wykorzystywane modele bazy danych	44
3.4.3 Translacje modeli	45
3.5 Zintegrowane serwisy	46

3.5.1	Jupyter Notebook	46
3.5.2	Jitsi Meet	47
3.5.3	QuizApp	48
3.5.4	Docker	49
3.6	Zasady działania ważniejszych elementów logiki biznesowej	50
3.6.1	Tworzenie kursu	50
3.6.2	Przebieg aktywności edukacyjnej	51
3.6.3	Tworzenie modułów lekcyjnych	54
3.6.4	Obsługa środowisk testowych	55
3.7	Szczegóły implementacyjne warstwy technicznej	57
3.7.1	Bezpieczeństwo systemu	57
3.7.2	Obsługa plików	60
3.7.3	Komunikacja między <i>Serwerem</i> , a <i>Orkiestratorem</i>	63
3.7.4	Komunikacja między <i>Aplikacją Webową</i> , a <i>Serwerem</i>	63
3.7.5	Obsługa maszyn wirtualnych	66
3.7.6	Przepływ danych w <i>Orkiestratorze</i> i schemat zarządzania serwisami	66
3.7.7	Motywacja i zasady działania mechanizmów niezawodności w <i>Orkiestratorze</i>	67
3.7.8	Integracja nowych serwisów	69
3.7.9	Przykład wykorzystania Ansible w celu integracji serwisu Jupyter	72
3.7.10	Obsługa wielu języków	74
3.8	Administracja systemem	74
3.9	Podsumowanie rozdziału	76
4	Organizacja pracy	77
4.1	Osoby w projekcie	77
4.2	Metodyka pracy	77
4.3	Etapy realizacji projektu	78
4.4	Wykorzystane praktyki i narzędzia	79
4.5	Podsumowanie rozdziału	83
5	Wyniki projektu	84
5.1	Prezentacja interfejsu użytkownika	84
5.1.1	Tworzenie modułów, kursów i przebieg aktywności edukacyjnych	84
5.1.2	Interfejs panelu administratora	93
5.1.3	Pozostałe widoki	97
5.2	Możliwości rozwoju	103
5.2.1	Nowe funkcjonalności	103
5.2.2	Poprawki implementacji aktualnego rozwiązania	103
5.3	Ocena rezultatów pracy oraz wnioski	104

Słownik

Aktywność edukacyjna (Zajęcia)

Zaplanowana przez nauczyciela aktywność dla uczniów, odbywająca się w określonym czasie, składająca się z zestawu wybranych wcześniej modułów.

Serwis

Oprogramowanie zewnętrzne, realizujące pewną funkcjonalność wspomagającą prowadzenie zajęć online np. Jitsi Meet - serwis do prowadzenia wideokonferencji.

Moduł

Gotowa do wykorzystania konfiguracja dostępnego serwisu pod dany dział tematyczny. Jest dodawana przez nauczyciela technicznego, a następnie wybierana przez nauczyciela, jako element aktywności edukacyjnej.

Model Driven Architecture (MDA)

Podejście do projektowania, opracowywania oraz wdrażania oprogramowania, które za pośrednictwem wytycznych dotyczących strukturyzowania specyfikacji oprogramowania wyrażonych przez modele (CIM, PIM, PSM) oddziela logikę aplikacyjną od technologii platformowych. Kolejne etapy transformacji MDA uszczególowiają model o coraz bardziej techniczne aspekty umożliwiające ostateczne uruchomienie środowiska zgodnie z określonymi wymaganiami, nie łącząc przy tym zdefiniowanych ograniczeń. Dzięki tak umożliwiionemu wysokopoziomowemu definiowaniu wymagań, osoby niezaznajomione z technicznymi detalami mogą konfigurować aktywności edukacyjne, mając pośredni wpływ na konfiguracje wykorzystywanych serwisów.

Computation Independent Model (CIM)

Model danych wykorzystywany w MDA, tworzony na podstawie wymagań wprowadzonych bezpośrednio przez użytkownika, którym w przypadku tworzenia zajęć jest nauczyciel. Zawiera dane niezależne od szczegółów technologicznych, takie jak moduły wykorzystywane podczas zajęć, liczba uczestników spotkania, wymagany czas spotkania itp.

Platform Independent Model (PIM)

Model danych wykorzystywany w MDA, tworzony na podstawie modelu CIM oraz zdefiniowanych przez administratora plików konfiguracyjnych znajdujących się w repozytorium. Reprezentuje generyczne informacje o środowisku, w którym zostanie uruchomiona dana aktywność edukacyjna np. ilość wymaganej pamięci RAM. Pozwala na utworzenie wielu różniących się szczegółami implementacyjnymi modeli PSM.

Platform Specific Model (PSM)

Model danych wykorzystywany w MDA, tworzony na podstawie modelu PIM z uwzględnieniem specyfikacji dostępnych maszyn wirtualnych. Zawiera szczegółowe dane takie jak typy maszyn, na których w określonym czasie zostają uruchomione odpowiednie serwisy umożliwiające prowadzenie zajęć.

Wstęp

W ostatnich latach nauka zdalna zaczęła odgrywać znaczącą rolę w sposobie prowadzenia spotkań o charakterze edukacyjnym. Z początku było to spowodowane obostrzeniami związanymi z pandemią, lecz z czasem okazało się być wygodną formą organizowania zajęć w niektórych dziedzinach nauki. Nagłe zapotrzebowanie na rozwój technologii umożliwiających przeprowadzanie internetowych lekcji spowodowało znaczny wzrost dostępnych na rynku narzędzi. Wiele z nich pozwala na wygodne monitorowanie w formie elektronicznej postępów studenta. Istnieją także serwisy pozwalające na odejście od standardowej formy prowadzenia zajęć zdalnych jaką jest wykład online, zastępując to bardziej interaktywną formą, która jest bliższa ćwiczeniom czy laboratoriom. Perspektywa wprowadzenia tych rozwiązań umożliwiających urozmaicenie sposobów przekazywania wiedzy uczniom wydaje się kusząca, lecz sam proces organizowania zajęć nie jest w żaden sposób ustandaryzowany, a korzystanie z niektórych technologii wychodzi często poza kompetencje nauczycieli szkół średnich czy nawet wyższych.

Niniejsza praca ma na celu zaprezentowanie nowego rozwiązania, ułatwiającego nauczycielowi przeprowadzanie zajęć online. Powstały system oparto na wzorcu Model Driven Architecture, a jego główną cechą jest zapewnienie użytkownikowi końcowemu czytelnego i intuicyjnego interfejsu pozwalającego na tworzenie i dołączanie do spotkań o określonej tematyce, bez eksponowania niepotrzebnych z jego punktu widzenia technicznych detali.

Efekt końcowy stanowi otwarte oprogramowanie (ang. open-source software) możliwe do wdrożenia w placówkach edukacyjnych, bądź korporacjach. Produkt wykorzystuje architekturę chmurową do udostępniania usług szerszemu gronu użytkowników. Prace skoncentrowano na wspomaganiu przeprowadzania zajęć technicznych, głównie informatycznych, jednak zaznaczono również możliwość wykorzystania systemu w zupełnie innych dziedzinach.

1. Cel prac i wizja produktu

Rozdział ma na celu przybliżenie problemu prowadzenia zajęć zdalnych oraz zaproponowanie jego rozwiązania.

Sekcja 1.1 przedstawia przykładowe przeszkody, z którymi musi zmierzyć się nauczyciel chcący poprowadzić wysokiej jakości zajęcia online. Sekcja 1.2 przedstawia analizę i porównanie dostępnych narzędzi usiłujących je rozwiązać, sekcja 1.3 podkreśla braki w rozwiązaniach konkurencyjnych oraz omawia dlaczego zaprojektowany przez nas system może znaleźć zastosowanie w środowiskach edukacyjnych. Sekcja 1.4 prezentuje i zwięzle uzasadnia sposób w jaki nasz produkt może rozwiązać postawiony przed nauczycielami problem, sekcja 1.5 omawia istniejący system o analogicznym przeznaczeniu bazujący na architekturze MDA i przybliża jego braki. Ostatecznie sekcja 1.6 przedstawia ryzyka z jakimi wiąże się tworzenie projektu zgodnie z przyjętymi przez nas założeniami.

1.1. Charakterystyka problemu

Bez większej trudności można zauważać różnicę w poziomie prowadzenia zdalnych zajęć edukacyjnych przez wyższe uczelnie techniczne, a pozostałe placówki edukacyjne. Powodem, z którego wynika istniejąca rozbieżność jest niezaznajomienie z zaawansowanymi technologiami, takimi jak notebooki, czy interaktywne serwisy do prowadzenia zajęć. W efekcie osoby nie korzystające z wyżej wymienionych technologii najczęściej ograniczają swoje zajęcia do prowadzenia zdalnych wykładów w formie telekonferencji.

Analizując rynek systemów zarządzania procesem nauczania (ang. Learning Management System) napotkamy wiele rozwiązań wspomagających prowadzenie zajęć w formie elektronicznej, z których każde posiada swój wyróżniający zestaw funkcjonalności. Cechami wspólnymi tych aplikacji są możliwości śledzenia postępów ucznia, takie jak przechowywanie informacji o ocenach, czy zaliczonych semestrach oraz zapewnienie środka komunikacji pomiędzy uczniem, a nauczycielem. Gdy rozważymy natomiast przydatność tych rozwiązań w dziedzinie organizowania zajęć, to najczęściej będzie ona ograniczona lub aplikacja nie będzie oferowała takich możliwości.

Nauczyciel chcąc przeprowadzić zajęcia z danej tematyki, najpierw musi wybrać narzędzie oferujące odpowiadającą funkcjonalność i poznać techniczne detale potrzebne do konfiguracji tego środowiska. Da się temu zaradzić wprowadzając odpowiedni poziom abstrakcji w interfejsie, z którym nauczyciel podejmuje interakcję, zapewniając bazową konfigurację, nieistotną z punktu widzenia samego przebiegu zajęć. Nie istnieje jednak obecnie na rynku rozwiązanie, które oferowałoby tak skonstruowany model interakcji z użytkownikiem i było jednocześnie łatwo rozszerzalne o nowe moduły.

Istotnym aspektem istniejących modułowych rozwiązań jest fakt, że są to rozwiązania komercyjne. Pociąga to za sobą głównie negatywne konsekwencje. Pierwszym i najbardziej oczywistym problemem jest koszt wykorzystania takiego oprogramowania - ze względu na niewielką konkurencję w tej dziedzinie, firmy mogą dyktować ceny, które nie będą w zasięgu budżetu każdej placówki edukacyjnej. Kolejnym źródłem problemów w przypadku korzystania z tych rozwiązań jest model zamkniętego oprogramowania (ang. closed-source software). Nawet w przypadku posiadania odpowiednio wykwalifikowanej kadry nauczycielskiej nie istnieje możliwość dowolnej ingerencji w działanie systemu, ze względu na usługowy model sprzedaży. Ogranicza to w znaczący sposób rozszerzalność tych rozwiązań o nowe moduły, a niekiedy sprawia, że nie jest to możliwe z perspektywy użytkownika.

1.2. Istniejące rozwiązania

Na rynku istnieje wiele narzędzi do przeprowadzania zajęć w formie zdalnej, lecz tylko niektóre z nich w zbliżony sposób realizują wymienione założenia. Na potrzeby niniejszej pracy dokonano analizy dostarczanych przez nie funkcjonalności, a następnie wybrano te aplikacje, którym najbliżej jest do rozwiązania postawionego wyżej problemu.

1.2.1. D2L Brightspace

D2L Brightspace jest komercyjną platformą edukacyjną przeznaczoną przede wszystkim do prowadzenia zajęć akademickich i szkoleń wewnętrz organizacji. Według informacji opublikowanych na stronie internetowej tego rozwiązania platforma integruje szacunkowo 1800 różnych narzędzi wspomagających nauczanie, a w tym moduły oparte o gry edukacyjne i quizy. Aplikacja pozwala na tworzenie własnych kursów z wykorzystaniem udostępnionych narzędzi. Istotnym atutem tego rozwiązania jest jego dostępność na urządzeniach mobilnych, co pozwala na udział w aktywnościach edukacyjnych nawet poza domem.

The screenshot displays the D2L Brightspace web interface. At the top, there's a header bar with the D2L Brightspace logo, the course name "Sample Course", and various user and system icons. Below the header is a navigation menu with links for Content, Assignments, Discussions, Quizzes, Students & Grades, Other Course Tools, and Help. On the left side, a sidebar shows a tree view of course modules: Overview, Bookmarks, Course Schedule, Table of Contents (24 items), Welcome! (3 items), Unit 1: Poetry (10 items), Unit 2: Non-Fiction (6 items), Unit 3: Presentations (3 items), and Your Last Day (2 items). There's also a button to "Add a module...". The main content area is titled "Table of Contents" and shows a "Welcome!" module with a thumbnail image of a coffee cup on a wooden surface. Below the image, there's a descriptive text for instructors and two buttons: "Upload / Create" and "Existing Activities".

Rysunek 1: Interfejs aplikacji D2L Brightspace

Użytkownik zainteresowany ofertą serwisu ma możliwość testowania go przez 30 dni. W trakcie tego okresu można zapoznać się z wyglądem i funkcjonalnościami kursu z perspektywy osoby, która go podejmuje, a także z samym jego tworzeniem. Podczas tworzenia kursu występuje możliwość dodawania aktywności edukacyjnych i dostosowywania modyfikowalnych modułów np. utworzenie własnej mapy gry. W przypadku chęci uzyskania pełnej funkcjonalności należy skontaktować się z firmą i negocjować indywidualną ofertę.

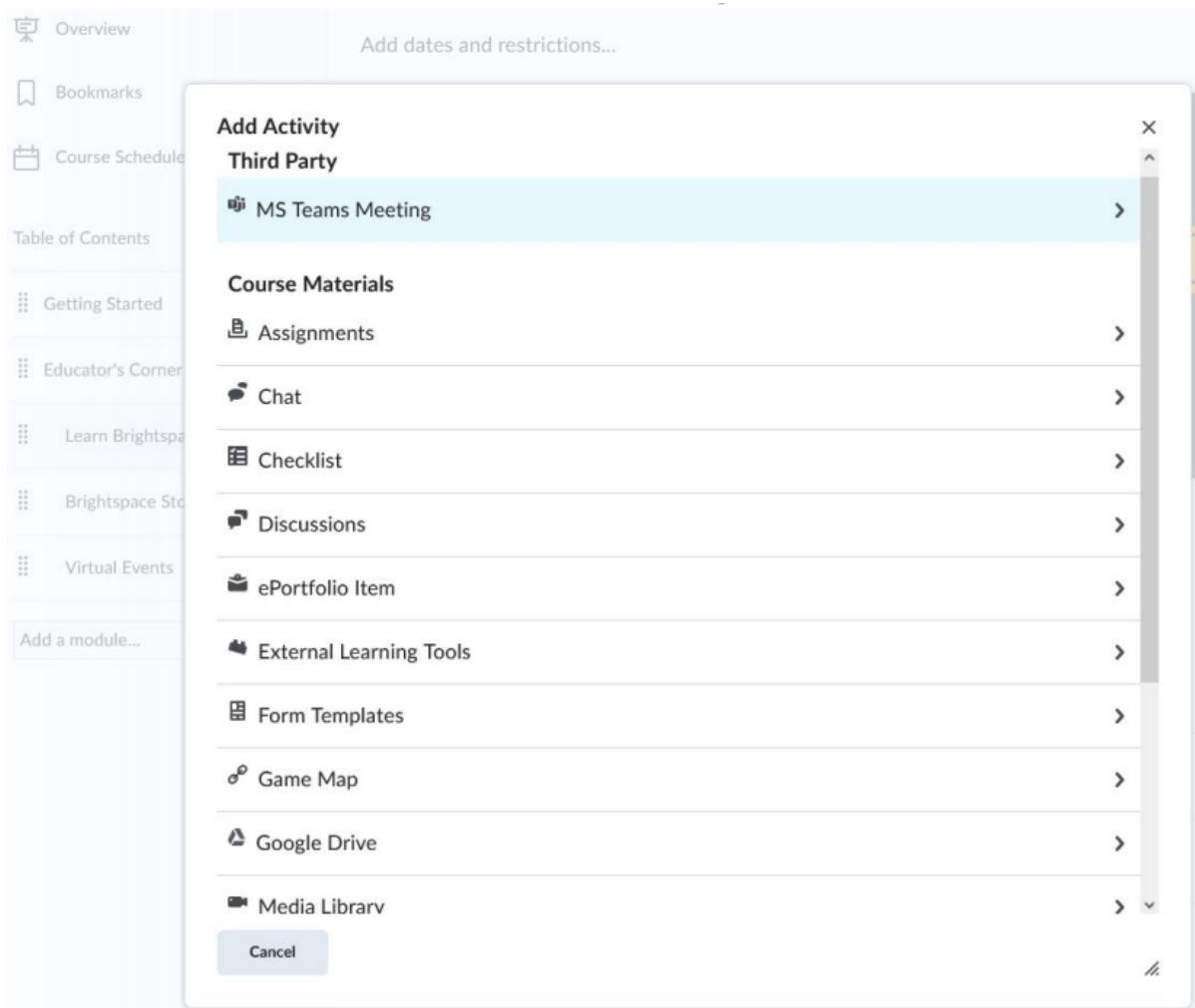
Interfejs w aplikacji jest intuicyjny i sprawia wrażenie nowoczesnego. Minimalistyczne ikony jasno reprezentują funkcje pełnione przez aktywne elementy strony, a pasek menu u góry ekranu pozwala na sprawne poruszanie się w obrębie serwisu.

Jedyną aktywnością dostępną podczas demo, którą można zakwalifikować jako gra edukacyjna jest mapa gry. Nauczyciel może na niej stawiać wizualne obiekty pełniące funkcję wyłącznie estetyczną oraz wyznaczyć ścieżkę przez aktywności edukacyjne - zajęcia i quizy. Ten sam efekt można uzyskać dodając te etapy bezpośrednio w kursie, tutaj mają one jedynie atrakcyjniejszą reprezentację wizualną.



Rysunek 2: Przykład tworzenia mapy gry - formy aktywności wspomaganej grą edukacyjną

Analizując zakres funkcjonalności, chcieliśmy sprawdzić jakość integrowanych przez serwis narzędzi edukacyjnych oraz ich zróżnicowanie. Podczas dodawania aktywności edukacyjnej na koncie ewaluacyjnym, istniała możliwość wyboru spośród jedynie około dwudziestu modułów. Nie jesteśmy pewni, czy jest to związane z wersją konta, czy twórcy D2L Brightspace inaczej definiują moduły. Zakładając, że jest to pełen zakres funkcjonalności, liczba ta roczarowuje w porównaniu do deklarowanej możliwości wykorzystania 1800 modułów edukacyjnych.



Rysunek 3: Lista wspieranych modułów edukacyjnych do wykorzystania w kursie

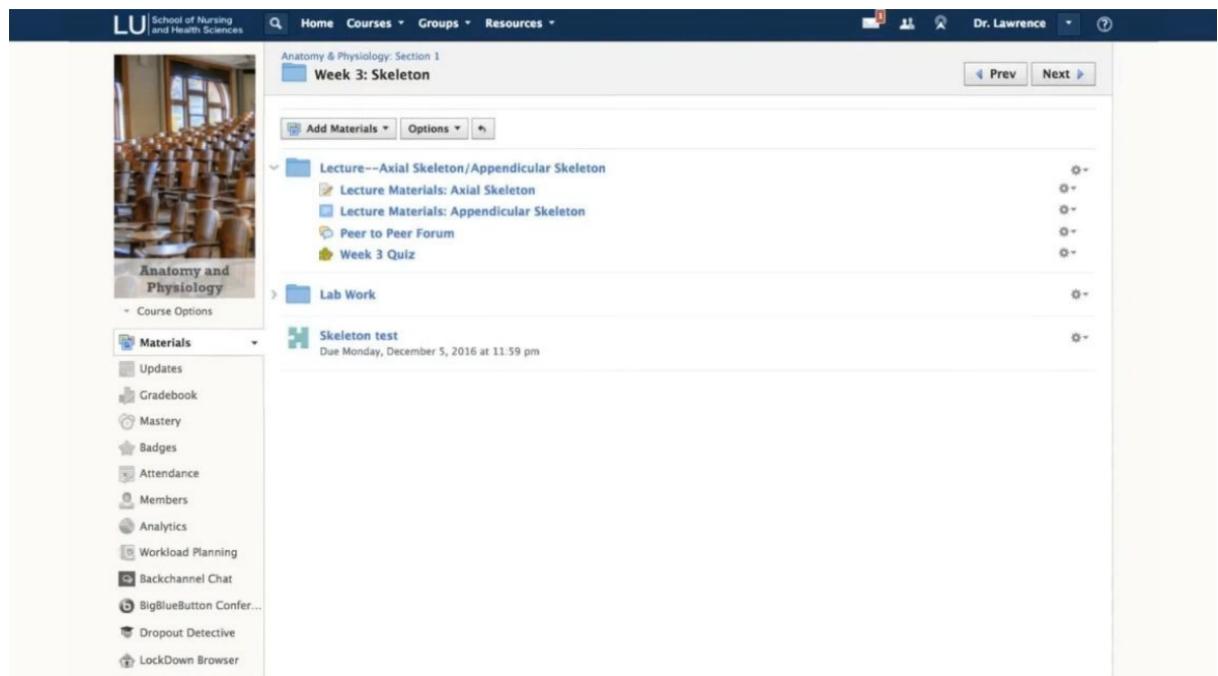
Ostatnim aspektem wartym poruszenia jest rozszerzalność systemu o nowe moduły. W przypadku D2L Brightspace jesteśmy całkowicie zdani na deweloperów aplikacji. Nie ma możliwości dodawania nowych modułów jako klient, co znacznie ogranicza funkcjonalność aplikacji i nie daje gwarancji, że pożądana integracja zostanie kiedykolwiek wprowadzona.

1.2.2. Schoology

Schoology jest komercyjną platformą edukacyjną nakierowaną na pomoc w realizowaniu programu nauczania K-12 (od wykształcenia podstawowego po średnie) obowiązującego w Stanach Zjednoczonych. System ten, podobnie do D2L Brightspace, również oferuje integrację zróżnicowanych modułów edukacyjnych i posiada podstawowe funkcjonalności LMS, takie jak tworzenie kursów, ocenianie uczniów, prowadzenie dyskusji.

W przeciwnieństwie do D2L Brightspace, Schoology nie oferuje darmowego okresu próbnego pozwalającego na zapoznanie się z aplikacją. Osobie prywatnej pozostaje jedynie analiza materiałów dostępnych w sieci, które są niestety często przedawnione. Klienci reprezentujący instytucje mogą ubiegać się o przeprowadzenie pokazu działania systemu, a następnie przejść do negocjacji oferty.

Interfejs nie jest mocną stroną tego rozwiązania. Istnieje ono na rynku od ponad dziesięciu lat, a szata graficzna sprawia wrażenie, jakoby nie została od tamtego momentu ani razu zaktualizowana. Brakuje w nim zdecydowanie przejrzystości, a załączone ikony wydają się rozmażane, przez co częściej znaleźć na stronie poszukiwaną funkcjonalność.



Rysunek 4: Interfejs Schoology

Na stronie Schoology możemy znaleźć pełną listę zintegrowanych modułów edukacyjnych, która liczbowo pokrywa się z deklaracjami. Zgodnie z założeniami systemu są to w większości aplikacje wspomagające nauczanie w programie K-12. Gdy przefiltrujemy je po przeznaczeniu, zaznaczając poziom akademicki, ich liczba spadnie do zaledwie kilku i będą to najczęściej zewnętrzne serwisy pokroju Google Drive.

The screenshot displays the Schoology App Center interface. At the top, there's a navigation bar with 'App Center' and a 'Become an App Developer' button. Below the navigation, there are filters for 'Type', 'Categories', 'Levels', 'Recommended for', and 'Most Popular'. A search bar is also present. The main area shows a grid of app cards:

- Edpuzzle**: Easily create beautiful interactive video lessons.
- Google Drive Resource App**: An application from Schoology for managing Google Drive files.
- Google Drive Assignments**: An application for assigning Google Docs, Spreadsheets, Slideshows, and Drawings.
- Nearpod**: An award-winning education technology company.
- McGraw Hill K-12 SSO**: A Single Sign-On app for McGraw Hill.
- PlayPosit (Individual/Basic)**: An online learning environment for creating interactive video lessons.

Rysunek 5: Schoology - Moduły edukacyjne

Rozszerzalność systemu o nowe moduły wygląda lepiej niż w przypadku analizowanego uprzednio D2L Brightspace, lecz dalej odbiega od poziomu swobody, który chcemy zaoferować w naszym rozwiążaniu. Na stronie App Center w aplikacji widnieje przycisk przekierowujący nas do rejestracji jako deweloper. Po dokonaniu rejestracji i udanej weryfikacji naszej aplikacji na stanowisko dewelopera, zostanie udostępniona nam możliwość integrowania własnych modułów z platformą Schoology.

1.2.3. Podsumowanie funkcjonalności

Po przeanalizowaniu rynku doszliśmy do wniosku, że nie istnieje darmowa aplikacja umożliwiająca organizowanie zajęć na podstawie dostarczanych modułów aplikacyjnych. Dostępne są rozwiązania płatne, które chwalą się mnogością zintegrowanych narzędzi do nauczania, lecz mają one problem z poszerzaniem swoich funkcjonalności, albo nie jest to możliwe, albo jest znacznie utrudnione przez konieczność interakcji z twórcami oprogramowania.

Cecha	Nasza aplikacja	D2l Brightspace	Schoology
Integracja z modułami edukacyjnymi	Tak	Tak	Tak
Rozszerzalność o nowe moduły	Tak	Zależne od deweloperów	Po rejestracji jako deweloper
Model sprzedaży	Darmowa	Płatna	Płatna
Otwarte oprogramowanie	Tak	Nie	Nie
Dostępny polski język	Tak	Nie	Nie

Tabela 1: Podsumowanie funkcjonalności

1.3. Motywacja projektu

Na podstawie analizy rozwiązań jesteśmy w stanie zauważyc wciąż istniejące zapotrzebowanie na produkt rozwiązujący problem organizacji zajęć online. W celu wypełnienia luki został stworzony system oferujący zarządzanie środowiskami do organizacji zajęć bez konieczności znajomości technicznych szczegółów oraz niewymagający instalacji i konfiguracji dodatkowego oprogramowania.

Tworzony produkt znacznie usprawni zarządzanie edukacją zdalną, tym samym zwiększać jakość oraz efektywność samej nauki. Dzięki nieskomplikowanemu oprogramowaniu po stronie użytkownika, zaprezentowana praca pozwoli osobom mniej zaznajomionym z najnowszymi rozwiązaniami na przekonanie się do korzystania z zaawansowanych technologii. Wpływ to korzystnie na ogólny poziom świadomości technologicznej pośród zarówno nauczycieli, jak i uczniów.

Utworzenie łatwo rozszerzalnego produktu o ogólnodostępnym kodzie źródłowym pozwoli stronom trzecim na udoskonalanie i rozbudowę naszego rozwiązania o funkcjonalności, których wstępnie nie przewidziano. Przykładowo uniwersytety informatyczne korzystające z produktu mogłyby jako tematy projektów programistycznych zlecać studentom tworzenie nowych, niewielkich modułów realizujących wymaganą przez nauczycieli funkcjonalność, gotowe moduły mogłyby być następnie prosto zintegrowane z naszym rozwiązaniem.

1.4. Wizja

Celem projektu jest stworzenie systemu usprawniającego planowanie oraz przeprowadzanie zajęć edukacyjnych w formie zdalnej. Zostanie to umożliwione poprzez automatyzację tworzenia różnorodnych środowisk edukacyjnych, szeroko wykorzystywanych podczas ćwiczeń w przestrzeni internetowej. Stworzenie takiego systemu znacząco usprawni pracę edukatorów, poprzez zauważalne skrócenie czasu potrzebnego do przygotowywania wszystkich niezbędnych programów i materiałów. Prowadzący zajęcia będą mogli zaplanować je z odpowiednim wyprzedzeniem, według własnych sprecyzowanych preferencji, a ogólnodostępne moduły będą zawierać wszystkie konieczne do przeprowadzenia zajęć materiały.

Docelowo produkt będzie wykorzystywany w placówkach edukacyjnych takich jak uniwersytety. Odpowiednio poinstruowany administrator skonfiguruje system umożliwiając dostęp wybranym użytkownikom, tj. nauczycielom oraz uczniom. Z uwagi na konieczność wysokiej

dostępności systemu oraz prostoty wdrożenia w placówkach nie posiadających rozbudowanej infrastruktury komputerowej wykorzystany zostanie zewnętrzny dostawca usług chmurowych.

Interfejs użytkownika stanowić będzie aplikacja webowa, dzięki której nauczyciele oraz uczniowie po zalogowaniu się na stronie będą mogli dołączyć do przygotowanych zajęć lekcyjnych, zaplanowanych uprzednio przez nauczyciela. Ponadto strona udostępnii standardowe funkcjonalności związane z przeprowadzaniem kursów takie jak dostęp do planów zajęć czy zrealizowanych na nich prac. Znaczna uwaga zostanie zwrócona na utworzenie rozwiązania możliwie prostego w obsłudze, udostępniającego wszelkie potrzebne instrukcje prowadzenia oraz uczestniczenia w zajęciach.

Moduły lekcyjne oraz instrukcje ich wykorzystania będą wprowadzane do systemu przez nauczycieli doświadczonych w korzystaniu z oprogramowania do nauczania online. Nauczyciele nie posiadający potrzebnych kwalifikacji będą mogli z nich korzystać w celu tworzenia nowoczesnych i bogatych w funkcjonalności kursów. Zastosowanie podejścia MDA zagwarantuje brak konieczności bycia zaznajomionym z wymaganiami serwisów, które będą wykorzystywane na zajęciach. Takie podejście pozwoli nauczycielom na zdefiniowanie wyłącznie wysokopoziomowych wymagań odnośnie aktywności edukacyjnych, które następnie zostaną automatycznie przetworzone na wymagania techniczne. Należy tutaj zaznaczyć, obie ze wspomnianych wyżej grup nauczycieli zyskają na korzystaniu z systemu, ponieważ będą mogły wielokrotnie wykorzystywać raz skonfigurowane moduły.

Zajęcia lekcyjne będą prowadzane z wykorzystaniem zewnętrznych serwisów, takich jak Jitsi Meet¹ czy Jupyter Notebook², które zostaną automatycznie uruchomione w chmurze chwilę przed planowanym rozpoczęciem zajęć. Serwisy będą w pełni funkcjonalne o wymaganej przez nauczyciela godzinie, a dostęp do nich będzie polegał na wejściu w dostarczone przez stronę internetową linki. Z uwagi na prostotę wykorzystania przez nauczycieli i uczniów, wszystkie serwisy będą dostępne z poziomu przeglądarki, a konfiguracja dostępu powinna ograniczyć się do uwierzytelnienia.

Serwisy, z pomocą których nauczyciele techniczni będą mogli tworzyć moduły, będą luźno powiązane (ang. loosely coupled) z systemem, co umożliwi łatwe rozszerzanie platformy o kolejne funkcjonalności jednocześnie nie komplikując interfejsu strony internetowej. Z uwagi na otwartość systemu nowe serwisy będą mogły być dodawane również przez odpowiednio wykwalifikowane osoby w miejscowościach, w których platforma zostanie wykorzystana. Skupimy się na wykorzystaniu narzędzi otwartych, zoptymalizowanych pod kątem udostępniania pojedynczej funkcjonalności, serwisy takie z uwagi na niewielką liczbę opcji powinny być proste w obsłudze dla nauczycieli i uczniów.

¹Serwis do prowadzenia wideokonferencji

²Serwis umożliwiający uruchamianie fragmentów kodu, wyświetlanie plików multimedialnych itp. w formacie popularnych notebooków

1.5. Prototyp

Niniejsze rozwiązanie stanowi rozwinięcie idei zaprezentowanej w pracy *On-demand provisioning of educational cloud-based services* [2], efektem której jest system o nazwie Cloud Orchestration [14], będący niekomercyjnym projektem studenckim pełniącym rolę "proof of concept" udowadniającym, że podejście MDA może mieć zastosowanie w organizacji zajęć online. Rozwiązanie skupiło się na wspomaganiu zajęć informatycznych, zaimplementowana została obsługa dwóch serwisów - Jupyter Notebook oraz Antidote³. Nauczyciel za pośrednictwem strony internetowej wybiera typ lekcji, temat oraz rozmiar serwisu. System bazujący na architekturze MDA wykorzystując te dane oraz informacje zdefiniowane przez osoby techniczne uruchamia w chmurze MCHE (Małopolska Chmura Edukacyjna) serwis wspomagający przeprowadzenie zajęć. Rysunek 6 przedstawia część interfejsu nauczyciela służącą utworzeniu zajęć.

Service Details	
Lessons Type	Lesson topic
<input checked="" type="radio"/> IT	<input type="radio"/> Networking
<input type="radio"/> Maths	<input type="radio"/> Cybersecurity
<input type="radio"/> Biology	<input type="radio"/> Programming
<input type="radio"/> Physics	<input checked="" type="radio"/> Operating Systems
	<input type="radio"/> Neural Networks
Service Size	Home directories
<input checked="" type="radio"/> Small	<input checked="" type="radio"/> MCHE
<input type="radio"/> Medium	<input type="radio"/> AGH
<input type="radio"/> Large	

Rysunek 6: Cloud Orchestration - Formularz tworzenia zajęć

Rozwiązanie przedstawione w projekcie Cloud Orchestration, mimo spełnienia założeń autora, ma istotne braki funkcjonalne, które nasza praca uzupełnia. Projekt składa się ze strony internetowej, pozwalającej jedynie na utworzenie aktywności edukacyjnej przez nauczyciela. Interfejsy ucznia, osoby mogącej tworzyć moduły czy też administratora nie zostały uwzględnione, a wszelkie powiadomienia były wysyłane drogą mailową. Istotnym ograniczeniem jest również możliwość wykorzystania zaledwie jednego serwisu do prowadzenia zajęć online, ponadto nie zostały zaimplementowane mechanizmy automatycznie zwalniające używane zasoby sprzętowe, przykładowo raz uruchomiony serwis Jupyter działał na maszynie wirtualnej do mo-

³Serwis do prowadzenia lekcji z wykorzystaniem powłoki systemu Linux

mentu jej wyłączenia lub ręcznej ingerencji administratora. Brak tych funkcjonalności przekreśla możliwość praktycznego wykorzystania systemu.

Prototyp wykorzystuje mniej nowoczesne i trudniejsze w potencjalnym utrzymaniu technologie (stos LAMP⁴ wraz z wieloma niespójnymi skryptami zaimplementowanymi w języku Python). Dodatkowo, niektóre problemy zostały tam rozwiązane w sposób uniemożliwiający generalizację. Przykładem są modele danych obsługiwane jak zwykłe pliki, zapisane lokalnie na maszynie, na której działał system. Zmiana struktury takiego pliku wymagałaby przejrzenia całego projektu w celu wyszukania miejsc, w których plik mógł zostać użyty. Z tych powodów nasz projekt nie rozwija dostępnego kodu źródłowego, jest on zaprojektowany i zaimplementowany od zera. Wzoruje się jedynie na ideach zaprezentowanych w prototype. Tabela 2 podsumowuje najważniejsze różnice pomiędzy naszym projektem, a prototypem.

Wspierane funkcjonalności	Nasza aplikacja	Prototyp
Uruchomienie pojedynczego serwisu w chmurze	Tak	Tak
Uruchomienie wielu serwisów w chmurze dla jednej lekcji	Tak	Nie
Interfejs webowy dla ucznia z planem zajęć i materiałami	Tak	Nie
Możliwość tworzenia modułów z poziomu przeglądarki	Tak	Nie
Informacje dotyczące zajęć dostępne na stronie	Tak	Nie
Zapisanie i dostęp do prac uczniów z zajęć	Tak	Nie

Tabela 2: Porównanie z prototypem

1.6. Analiza zagrożeń

Tworzenie opisanego systemu niesie ze sobą niewątpliwie wiele potencjalnych zagrożeń, w większości przypadków spowodowanych trudnymi do przewidzenia wydarzeniami.

Opieranie głównych funkcjonalności na niezależnych serwisach powiązane jest z ryzykiem, w którym przestają one być ogólnodostępne. Najważniejszym zadaniem projektu jest umożliwienie nauczycielom prowadzenia zajęć z wykorzystaniem dostępnych w internecie narzędzi. Możemy więc wyobrazić sobie przypadek, w którym korzystanie z danego serwisu, na którym oparte zostało wiele modułów, staje się niemożliwe. Wskutek tego dotychczasowe prace wykonane nad jego integracją zostają pozbawione sensu i bezpowrotnie utracone.

Kolejnym istotnym ryzykiem wartym rozpatrzenia jest nieodpowiedni wybór technologii wykorzystywanych do realizacji projektu. Może okazać się, że obiecujące i dobrze znane narzędzia nie spełniają wszystkich wymagań. W takim przypadku kontynuacja pracy wiąże się z dodatkowym nakładem czasu oraz potrzebą poszukiwania alternatywnych rozwiązań w celu pozbicia się powstały problemów. Szczególnie ważnym wydaje się być wybór odpowiedniego środowiska chmurowego, pełniącego kluczową rolę w systemie. Środowisko takie może nie dostarczać wszystkich niezbędnych do poprawnego działania projektu funkcjonalności, wskutek czego nieunikniona staje się potrzeba szukania rozwiązania zastępczego.

By zminimalizować ryzyko wystąpienia wymienionych zagrożeń, konieczne jest wcześniej zaplanowanie pracy oraz dokładna analiza dostępnych serwisów i technologii. Umożliwia to weryfikację udostępnianych przez nie funkcjonalności oraz pozwala na określenie ich przydatności w tworzonym projekcie. Regularne sprawdzanie informacji na stronach domowych integrowanych serwisów minimalizuje ryzyko nagłej utraty możliwości korzystania z któregoś

⁴Linux Apache MySQL PHP - zbiór technologii kiedyś powszechnie wykorzystywanych do budowania aplikacji internetowych

z nich i daje więcej czasu na znalezienie zamiennika. Wykonanie wcześniejszej organizacji za- dań i estymacji czasu redukuje możliwość niepowodzenia w późniejszych fazach projektu, co z kolei pozwala na zrealizowanie ustalonych założeń pracy i dostarczenie wymaganych funkcjonalności.

1.7. Podsumowanie rozdziału

W rozdziale przybliżono problem prowadzenia zajęć zdalnych, który nasz produkt ma za za- danie rozwiązać. Pokazano już istniejące podejścia do problemu, zarówno konkurencyjne, jak i sam prototyp systemu. Określono braki tych rozwiązań względem oczekiwanej efektu oraz przedstawiono wizję, która będzie podstawą proponowanych w drugim rozdziale funkcjonalno- ści i podziału na podsystemy. Następnie w trzecim rozdziale zostanie pokazana jej realizacja.

2. Zakres funkcjonalności

Rozdział prezentuje aktorów oraz funkcjonalności, którymi cechuje się utworzony przez nas system.

Sekcja 2.1 przedstawia aktorów wchodzących w interakcję z systemem. Sekcja 2.2 wskazuje historie użytkowników (ang. user stories), sekcja 2.3 porządkuje zebrane wymagania i podsumowuje funkcjonalności składowych systemu. Sekcja 2.4 specyfikuje i uzasadnia wymagania niefunkcjonalne. Sekcja 2.5 prezentuje wstępny podział na systemy odpowiadające za realizację wymaganych funkcji.

2.1. Przedstawienie aktorów

W interakcję z systemem może wchodzić czterech aktorów, są to: Nauczyciel techniczny, Nauczyciel, Student oraz Administrator.

- **Nauczyciel techniczny** - osoba tworząca moduł, która musi znać technologię potrzebną do wykorzystania danego serwisu oraz potrafić samodzielnie skonfigurować ją w celu przygotowania zajęć. Nie jest głównym beneficjentem aplikacji, ale jego pomoc jest podstawą odpowiedniego działania systemu i umożliwia korzystanie z niego nauczycielom oraz uczniom. Jednocześnie jako nauczyciel techniczny, osoba posiada wszelkie uprawnienia przypisane nauczycielowi.
- **Nauczyciel** - jeden z głównych odbiorców naszego systemu. Osoba, której w głównej mierze ma on ułatwić prowadzenie zdalnych zajęć i dać możliwość przeniesienia ich na nowy, wcześniej nieosiągalny z powodu bariery technologicznej poziom. Osobami przypisanymi do tej roli mają być nauczyciele szkół średnich oraz wyższych, chcący prowadzić lekcje z wykorzystaniem najnowszych stworzonych w tym celu narzędzi, bez konieczności znajomości używanych przez nie technologii.
- **Uczeń** - osoba, która ma znacząco zyskać na wykorzystaniu naszej aplikacji zarówno dzięki podniesieniu jakości nauczania zdalnego, jak i poznaniu nowych technologii do tego wykorzystywanych. Za pośrednictwem jednej strony internetowej będzie mógł korzystać z niedostępnej dotychczas formy zdalnych ćwiczeń/laboratoriów.
- **Administrator** - aktor posiadający możliwość największego ingerowania w działanie samego systemu. Jest on niezbędny do wdrożenia i utrzymania aplikacji, działa na poziomie placówki lub zbioru placówek. Nadzoruje działanie systemu oraz ma możliwość naprawiania potencjalnych błędów związanych ze złym jego wykorzystaniem.

2.2. User Stories

Na podstawie rozmów z klientem oraz analizy problemu zostały zanotowane oraz przypisane do odpowiednich aktorów historie użytkownika.

Jako nauczyciel techniczny:

- T1.** Chcę móc skonfigurować i udostępnić moduł lekcyjny, aby umożliwić nauczycielom wykorzystanie wysokiej jakości materiałów bez konieczności samodzielnej konfiguracji.
- T2.** Przy tworzeniu modułu chcę mieć możliwość wyboru spośród wielu dostępnych serwisów, żeby lepiej dostosować technologie do zastosowania.
- T3.** Chcę móc dodać instrukcję do korzystania z modułu, żeby nie musieć wielokrotnie tłumaczyć nauczycielom jak z niego korzystać.
- T4.** Chcę mieć możliwość podglądu i edycji utworzonych w przeszłości modułów, żeby móc je zaktualizować w związku z rozwojem pewnych technologii.
- T5.** Chcę mieć możliwość przetestowania modułu, żeby upewnić się, że poprawnie go skonfigurowałem.
- T6.** Chcę mieć możliwość ustawienia modułu jako prywatny, żeby tylko ja mógł z niego skorzystać.
- T7.** Chcę móc zobaczyć kto używa utworzonych przeze mnie modułów i w jakich kursach.
- T8.** Chcę mieć dostęp do instrukcji dotyczących sposobu konfiguracji wybranego serwisu, żeby wiedzieć jak to zrobić poprawnie.

Jako nauczyciel:

- N1.** Chcę móc utworzyć kurs wykorzystując gotowe moduły lekcyjne, żeby nie musieć ich uprzednio konfigurować.
- N2.** Chcę móc skorzystać z dopracowanych modułów lekcyjnych, żeby zwiększyć jakość swoich zajęć.
- N3.** Chcę móc przypisać określonych uczniów do kursu, aby uzyskali oni dostęp do prowadzonych przeze mnie zajęć.
- N4.** Chcę móc zaplanować aktywności edukacyjne na określone dni i godziny, aby prowadzony przeze mnie kurs trzymał się ustalonego harmonogramu.
- N5.** Chcę mieć możliwość wykorzystania wideokonferencji jako jednego z serwisów, ponieważ jest mi potrzebna do przeprowadzenia większości zajęć.
- N6.** Podczas zajęć chcę móc wykorzystać wszystkie wymagane zewnętrzne serwisy za pośrednictwem przeglądarki, żeby móc przeprowadzić zajęcia bez potrzeby pobierania dodatkowego oprogramowania.
- N7.** Chcę mieć dostęp do instrukcji wykorzystania modułów, żeby nie musieć szukać zewnętrznych pomocy.
- N8.** Chcę mieć łatwy dostęp do wszystkich informacji potrzebnych do rozpoczęcia zajęć, żeby nie musieć ich długo szukać.
- N9.** Chcę mieć dostęp do szczegółów wszystkich utworzonych przeze mnie kursów, żeby móc wydajnie prowadzić zajęcia dla różnych grup uczniów.

- N10.** Chcę móc zobaczyć prace, które uczniowie wykonali na zajęciach, żeby kontrolować ich postępy.
- N11.** Chcę mieć dostęp do planu zajęć, żeby o nich nie zapomnieć.
- N12.** Chcę móc edytować daty, godziny oraz szczegóły niektórych spotkań, żeby dostosować je do niespodziewanych czynników zewnętrznych.
- N13.** Chcę mieć możliwość podglądu przeprowadzonych w przeszłości spotkań, żeby monitorować postęp kursu.
- N14.** Chcę móc otrzymywać powiadomienia związane z danym kursem, żeby być w stanie szybko zareagować na zdarzenia zewnętrzne.
- N15.** Chcę móc przetestować moduł przed wybraniem go do aktywności edukacyjnej, żeby mieć pewność, że spełnia on moje oczekiwania.

Jako uczeń:

- U1.** Chcę mieć dostęp do kursów, do których jestem przypisany, żeby móc brać udział w zajęciach.
- U2.** Chcę mieć wszystkie informacje potrzebne do wzięcia udziału w zajęciach, żeby nie musieć pytać o pomoc nauczyciela.
- U3.** Podczas zajęć chcę móc wykorzystać wszystkie wymagane zewnętrzne serwisy za pośrednictwem przeglądarki, żeby być w stanie dołączyć do zajęć bez potrzeby pobierania dodatkowego oprogramowania.
- U4.** Chcę mieć dostęp do planu zajęć i otrzymywać powiadomienia o nadchodzących zajęciach, żeby ich nie przegapić.
- U5.** Chcę mieć możliwość zapisu niektórych materiałów z różnych kursów, żeby mieć najważniejsze rzeczy zebrane w jednym miejscu.
- U6.** Chcę żeby moje prace z zajęć mogły być dostępne dla nauczyciela, abym nie musiał ich wysyłać ręcznie innymi środkami.

Jako administrator:

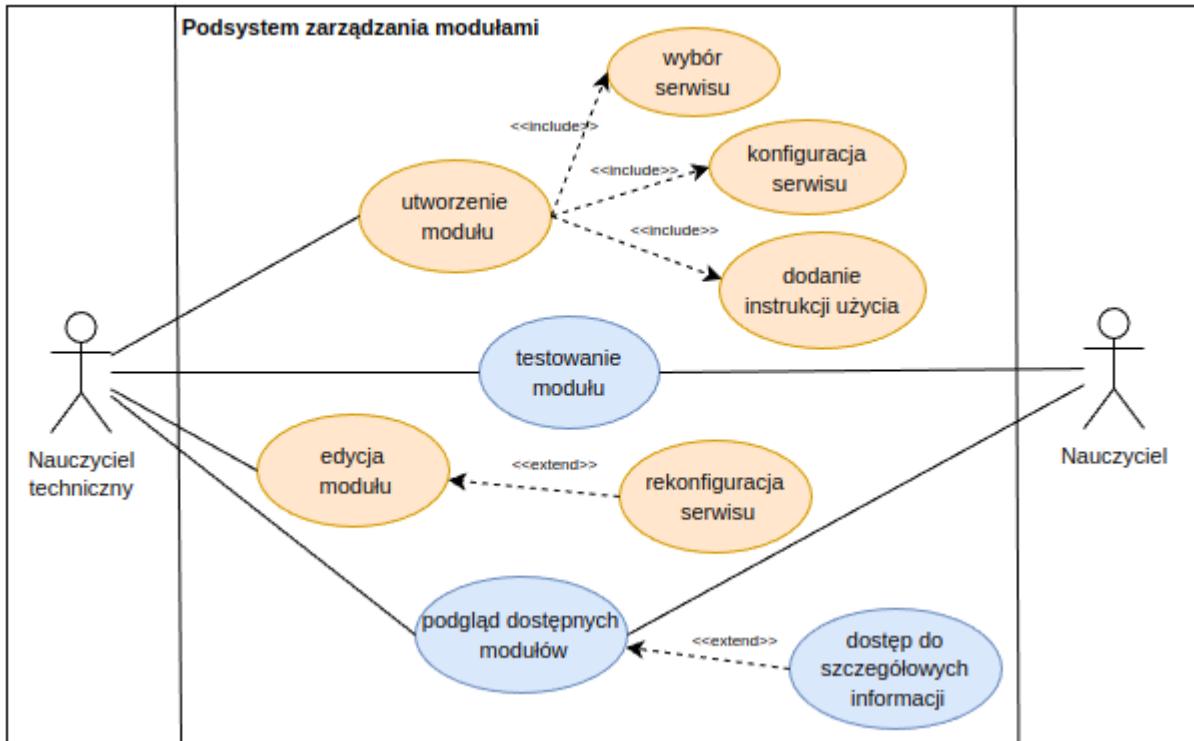
- A1.** Chcę móc tworzyć konta użytkowników oraz przypisywać im role w systemie, żeby dane były spójne z danymi obowiązującymi w mojej placówce.
- A2.** Chcę mieć możliwość zarządzania zaplanowanymi aktywnościami edukacyjnymi, żeby w razie problemów pomóc innym użytkownikom.
- A3.** Chcę móc uruchomić system w placówce z ograniczonymi zasobami sprzętowymi, żeby nie generować dodatkowych kosztów związanych z kupnem i utrzymaniem infrastruktury.
- A4.** Chcę mieć dostęp do danych systemu, takich jak ilość wykorzystanych zasobów, żeby móc monitorować jego stan.
- A5.** Chcę móc definiować polityki w systemie, żeby zarządzać dostępem do zasobów przez poszczególnych nauczycieli.
- A6.** Chcę móc konfigurować techniczne detale systemu, żeby zapewnić jak najlepszy poziom świadczonych usług.

2.3. Grupy funkcyjonalne

Wykorzystując historie użytkownika możemy pogrupować wymagania na trzy podsystemy funkcyjonalne dotyczące kolejno modułów (sekcja 2.3.1), kursów i zajęć (sekcja 2.3.2) oraz administracji (sekcja 2.3.3). Każda funkcyjonalność podsystemu zawiera odniesiki do historii użytkowników, na podstawie których została dodana. Dla wizualizacji i ułatwienia odbioru dla każdego z podsystemów przygotowano osobny diagram przypadków użycia (ang. Use Case diagram).

2.3.1. Podsystem zarządzania modułami:

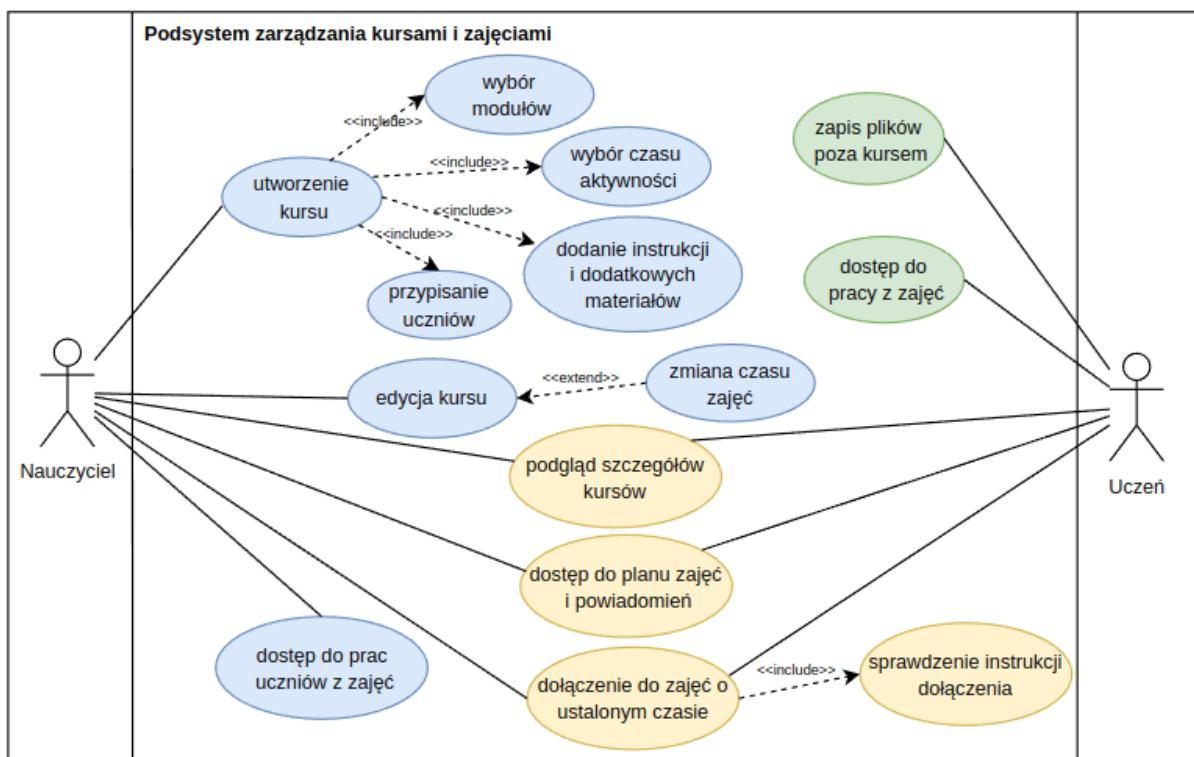
- wybór spośród wielu dostępnych serwisów przy tworzeniu modułu (T2);
- możliwość konfiguracji serwisu, który ma być wykorzystany w module(T1);
- dostęp do instrukcji konfiguracji serwisów (T8);
- dodanie instrukcji korzystania z modułu (T3);
- opcja edycji modułu z możliwością rekonfiguracji serwisu (T4);
- podgląd dostępnych modułów (T4, N1);
- dostęp do szczegółowych danych modułu (T4, T7, N7);
- opcja ograniczenia dostępu do modułu (T6);
- możliwość przetestowania modułu (T5, N15).



Rysunek 7: Podsystem zarządzania modułami - Use Case diagram

2.3.2. Podsystem zarządzania kursami i zajęciami:

- tworzenia kursu z wykorzystaniem ogólnodostępnych modułów (N1);
- przypisanie uczestników do kursu (N3);
- wybór dat i godzin odbywania się zajęć wchodzących w skład kursu (N4);
- dołączenie do zajęć z przeglądarki o ustalonym czasie (N6, U3);
- podgląd szczegółów kursów (N9, N13, U1);
- dostęp do instrukcji rozpoczęcia zajęć (N8);
- dostęp do instrukcji dołączenia do zajęć (U2);
- edycja zawartości i konfiguracji kursu (N12);
- podgląd planu zajęć (N11, U4);
- powiadomienia dotyczące kursu (N14, U4);
- dostęp do prac uczniów z zajęć (N10, U6);
- możliwość zapisu w punkcie centralnym materiałów z różnych kursów (U5).

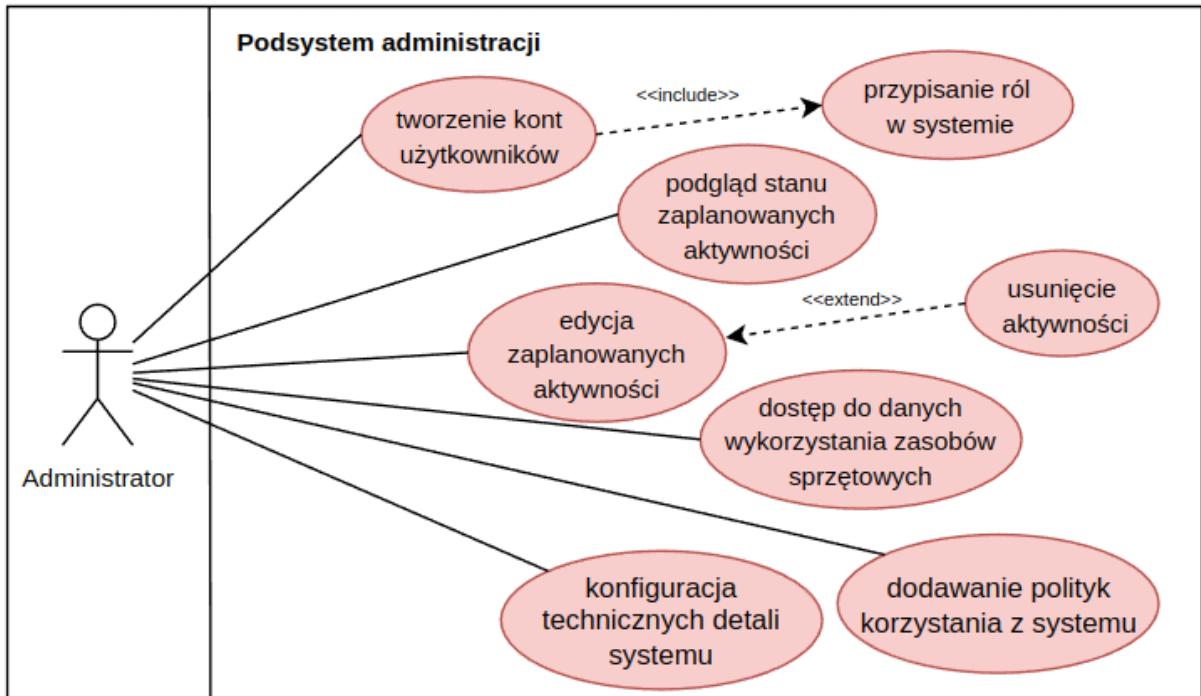


Rysunek 8: Podsystem zarządzania kursami i zajęciami - Use Case diagram

2.3.3. Podsystem administracji:

- tworzenie kont użytkowników (A1);
- przypisywanie ról użytkownikom (A1);
- podgląd stanu zaplanowanych aktywności edukacyjnych (A2);
- edycja szczegółów zaplanowanych zajęć (A2);

- powiadomienie nauczyciela o zmianie szczegółów zajęć (A2);
- dostęp do danych wykorzystania zasobów sprzętowych (A4);
- dodawanie polityk korzystania z systemu (A5);
- konfigurowanie technicznych detali systemu (A6).



Rysunek 9: Podsystem administracji - Use Case diagram

2.4. Specyfikacja wymagań niefunkcjonalnych

Biorąc pod uwagę historie użytkownika, przedstawione wymagania funkcjonalne oraz rozważając charakter środowiska, w którym będzie działał system, możemy wyszczególnić następujące wymagania niefunkcjonalne:

a) System ma bazować na wzorcu MDA.

Praca ma sprawdzić, czy wzorzec MDA może mieć zastosowanie w środowiskach edukacyjnych spełniając przy tym uprzednio zdefiniowane wymagania. Prototyp wspomniany w sekcji 1.5 udowodnił jedynie, że bazując na tej architekturze da się stworzyć system przekształcający wysokopoziomowe wymagania nauczyciela na implementację działających serwisów, nie pokazał jednak, czy rozwiązanie jest skalowalne i nadające się do wykorzystania w rzeczywistości.

b) Aplikacja musi być wysoce dostępna, bez częstych awarii, spadków wydajności lub długotrwałych przerw na prace techniczne.

Jest to związane z realiami panującymi na uczelniach, plany zajęć są często układane w sposób mocno utrudniający przekładanie aktywności na inne dni i godziny. Ponadto potencjalne problemy z dostępem do systemu w godzinach zajęć mogłyby być bardzo stresujące zarówno dla nauczycieli jak i uczniów chcących dołączyć do zajęć.

c) Dane muszą być przechowywane w sposób bezpieczny.

Aplikacja będzie operować na prywatnych danych takich jak prace uczniów wykonane na zajęciach, indeksy studentów czy też adresy e-mail. Potencjalne wycieki danych mogłyby mieć istotne konsekwencje zarówno dla organizacji korzystających z naszego systemu jak i indywidualnych użytkowników.

d) System musi być możliwy do uruchomienia w placówkach nieposiadających rozbudowanej infrastruktury sprzętowej.

Wymaganie wiąże się ponownie z realiami większości firm i placówek edukacyjnych, organizacje nie mają wystarczających funduszy na kupno sprzętu oraz utrzymanie personelu niezbędnego do nadzorowania jego działania.

e) Wszystkie zewnętrzne serwisy muszą być gotowe do użytku o zaplanowanej godzinie.

Jako że zastosowaniem produktu jest prowadzenie zajęć online, wszystkie serwisy potrzebne do ich realizacji muszą być gotowe do użytku w momencie rozpoczęcia zajęć. Konieczne jest więc uruchamianie ich przed rozpoczęciem danej aktywności edukacyjnej, tak by użytkownicy nie doświadczyli opóźnienia związanego z proceseminicjalizacji serwisu.

f) Wszystkie zewnętrzne serwisy muszą być dostępne za pośrednictwem przeglądarki internetowej.

Główna idea projektu skupia się na prostym użytkowaniu serwisów edukacyjnych. Jedynym powszechnym i przenośnym oprogramowaniem spełniającym to założenie jest przeglądarka internetowa. Serwisy powinny być możliwe do wykorzystania za jej pośrednictwem po podaniu linków oraz ewentualnym prostym wypełnianiu formularzy zgodnie z instrukcjami podanymi przez nauczyciela technicznego.

2.5. Współpracujące podsystemy

Stworzony system składa się z kilku współpracujących ze sobą elementów. Użytkownik podejmuje interakcję tylko z aplikacją webową, a pozostałe podsystemy odpowiadają za logikę działania aplikacji. Wspomniane elementy to:

- **Aplikacja Webowa** - jedyna część systemu z którą bezpośrednią styczność mają nauczyciele (zarówno zwykli jak i techniczni) oraz uczniowie. Nauczyciel za pośrednictwem intuicyjnego interfejsu może stworzyć kurs składający się z opartych na modułach aktywności edukacyjnych, a następnie przypisać do niego wybranych uczniów. Zajęcia będą się odbywać według ustalonego wcześniej harmonogramu, a dostęp do nich zostanie umożliwiony poprzez dostarczone linki. Dodatkowo aplikacja umożliwia uczniowi oraz nauczycielowi dostęp do prac wykonanych podczas zajęć. Stworzone zostały także dedykowane widoki dla nauczyciela technicznego, dzięki którym może dodawać oraz testować nowe moduły. Aplikacja udostępnia również większość funkcjonalności potrzebnych administratorowi do zarządzania systemem. Wspomniane wyżej zadania mogą być zrealizowane dzięki komunikacji tego podsystemu z back-endem.
- **Back-end** - Część odpowiadająca za organizację logiki działania systemu i zajmująca się obsługą żądań otrzymanych od Aplikacji Webowej. Umożliwia przechowywanie dodawanych przez nauczycieli technicznych modułów. Pozwala również na przetworzenie prośby przychodzącej od nauczyciela odnośnie stworzenia nowych zajęć i przygotowuje

spotkanie uwzględniając wszystkie warunki oraz dobierając odpowiednie narzędzia. Podsystem dostarcza wszelkie niezbędne mechanizmy kontroli dostępu do zasobów.

Back-end odpowiada także za interakcję z trzecią częścią systemu, czyli serwisem chmurowym, który umożliwia skorzystanie z narzędzi wybranych przez nauczyciela do prowadzenia zajęć.

- **Serwis Chmurowy** - Traktowany w naszym systemie jako dostawca infrastruktury potrzebnej do prowadzenia zajęć zdalnych oraz przechowywania plików takich jak prace uczniów lub konfiguracje serwisów. Umożliwia korzystanie z aplikacji placówkom nie posiadającym rozwiniętej architektury informatycznej. Pozwala także na optymalizację kosztów, ponieważ zaraz po zakończeniu zajęć zasoby sprzętowe są zwalniane.

2.6. Podsumowanie rozdziału

W rozdziale skoncentrowano się na przedstawieniu aktorów oraz przyporządkowaniu im historii użytkownika. Funkcjonalności powstałe na ich podstawie zebrane w grupy funkcjonalne oraz zaprezentowano w postaci diagramów przypadków użycia. W trzecim rozdziale zostaną rozwinięte zagadnienia w nich zawarte wykorzystując między innymi diagramy sekwencji. Jako rozszerzenie opisu architektury zawartego w sekcji 2.5 zostaną omówione szczegółowe i techniczne elementy systemu. Przedstawione wymagania niefunkcjonalne stanowią podstawę zaprezentowanych w rozdziale trzecim decyzji projektowych.

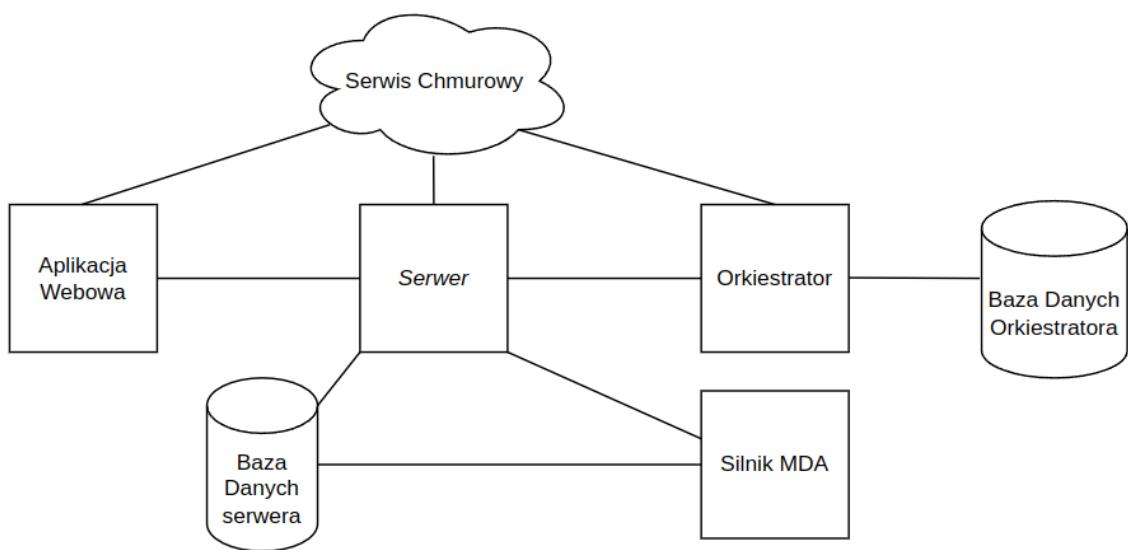
3. Wybrane aspekty realizacji

Rozdział opisuje działanie systemu, przyjęte założenia i wykorzystane technologie. Omawia w jaki sposób zostały spełnione postawione przed systemem wymagania funkcjonalne i niefunkcjonalne.

Sekcja 3.1 przedstawia zgodny z zaimplementowaną wersją podział na podsystemy, omawia dokładne zadania każdego komponentu oraz obrazuje architekturę całego systemu. Sekcja 3.2 opisuje stos technologiczny wybrany do implementacji projektu. Sekcja 3.3 pokazuje model danych, na którym operują podsystemy, następnie krótko opisuje co oznaczają jego wybrane elementy. Sekcja 3.4 dokładniej omawia podejście MDA i pokazuje jak zostało wykorzystane w naszym projekcie. Sekcja 3.5 opisuje zewnętrzne serwisy zintegrowane z systemem oraz wymienia funkcjonalności jakich dostarczają. Sekcja 3.6 bazując na wprowadzonych wcześniej pojęciach przedstawia zasady działania najważniejszych funkcjonalności systemu. Sekcja 3.7 opisuje techniczne aspekty implementacyjne ważniejszych elementów systemu, wprowadza również pojęcia niezbędne do prawidłowego administrowania systemem, które omówiono w sekcji 3.8.

3.1. Struktura systemu

W sekcji 2.5 wstępnie zaproponowano podział systemu na podsystemy umożliwiające spełnienie wymagań funkcjonalnych jak i niefunkcjonalnych oczekiwanych od końcowego produktu. Z uwagi na jego wysoką złożoność konieczne jest wprowadzenie bardziej drobnoziarnistego podziału na komponenty, z których każdy będzie spełniał jasno zdefiniowany zbiór zadań. Poniżej przedstawiono podział, w którym podsystemy odpowiadają tym rzeczywiście zaimplementowanym i wykorzystywanym w projekcie. W kolejnych sekcjach zostanie doprecyzowane w jaki sposób zaprojektowana została interakcja między nimi, po czym opisane i uzasadnione zostaną dokładne decyzje dotyczące wybranych technologii umożliwiających implementację projektu.



Rysunek 10: Architektura systemu

Architektura systemu może zostać w sposób wysoko-poziomowy pokazana na diagramie 10. Połączenia między komponentami oznaczają zależności komunikacji, należy jednak za-

znaczyć, że jest to podział logiczny i nie wszystkie podsystemy będą musiały komunikować się przez sieć.

3.1.1. Serwer

Utworzony produkt jest aplikacją internetową, kluczowym elementem w implementacji każdej takiej aplikacji jest serwer webowy. W dalszej części pracy odwołujemy się do niego jako *Serwer*.

Głównym zadaniem *Serwera* jest pełnienie roli pośrednika między aplikacją internetową, bazą danych oraz pozostałymi podsystemami. Jest to jedyny z wewnętrznych podsystemów, który będzie bezpośrednio obsługiwał żądania użytkowników, kluczowe jest zatem żeby dostarczał on wszystkich niezbędnych funkcjonalności związanych z bezpieczeństwem oraz uwierzytelnianiem, zostało to szerzej opisane w sekcji 3.7.1. Wśród jego obowiązków znajduje się także administrowanie kontami użytkowników oraz ich rolami i politykami w systemie.

Powołując się na wymagania funkcjonalne konieczne było, żeby *Serwer* obsługiwał zarządzanie kursami oraz aktywnościami edukacyjnymi tworzonymi przez nauczycieli, jednak z uwagi na stopień skomplikowania tego zadania, część funkcjonalności została wydzielona do niżej opisanych podsystemów. Po stronie *Serwera* w tym zakresie zostało umożliwienie dokonywania typowych operacji CRUD⁵ wraz z walidacją wszelkich akcji użytkowników.

Utworzenie aktywności edukacyjnych polega na wyborze spośród dostępnych modułów lekcyjnych. Kolejnym zadaniem *Serwera* jest umożliwienie nauczycielom technicznym zarządzania tymi modułami. Ważną funkcjonalnością, udostępnianą przez *Serwer*, jest także dopuszczenie tworzenia środowisk testowych, aby nauczyciel mógł sprawdzić czy wybrany moduł spełnia jego wymagania dla danej aktywności oraz żeby nauczyciel techniczny mógł się upewnić czy wprowadzona konfiguracja działa poprawnie.

Żeby można było tworzyć moduły konieczna jest w systemie obecność serwisów, które moduły mogą wykorzystywać. Kolejnym zadaniem *Serwera* jest umożliwienie konfigurowania takich serwisów w sposób generyczny, w sekcji 3.7.8 została dokładnie opisana realizacja tego celu.

Serwisy są uruchamiane na maszynach wirtualnych dostępnych w systemie dostawcy usług chmurowych. Uruchamianie maszyn wiąże się z kosztami, różnymi w zależności od maszyny, z tego powodu niedopuszczalne jest zezwolenie systemowi na automatyczny wybór spośród dowolnych maszyn. Konfiguracja jakie maszyny mogą być używane przez system jest jednym z zadań administratora. Zostało ono umożliwione za pośrednictwem *Serwera*. Uniezależnia to również w pewnym stopniu cały system od wyboru dostawcy usług chmurowych i pozwala na wybór innego.

Użytkownicy aplikacji mogą w różnych celach przesyłać pliki, przykładowo pliki konfiguracyjne serwisów czy też pliki skojarzone z pojedynczymi aktywnościami edukacyjnymi. Dodatkowo konieczne jest zapisywanie i umożliwienie odczytu plików z pracą zrealizowaną przez uczestników kursu podczas zajęć. Zadaniem *Serwera* jest zarządzanie plikami, w szczególności przyznawanie dostępu i przypisywanie plików do poszczególnych użytkowników. Samo przechowywanie i ostateczne zabezpieczenie plików oddelegowane zostało do dostawcy usług chmurowych, opisanego w jednej z kolejnych sekcji.

⁵create, read, update, delete - czyli zbiór podstawowych funkcji w aplikacjach korzystających z baz danych

3.1.2. Orkiestrator

Podsystemem, do którego została wydzielona odpowiedzialność zarządzania cyklem życia instancji maszyn wirtualnych oraz uruchomionych na nich serwisów jest *Orkiestrator*. Decyzja o wydzieleniu go ze struktury *Serwera* została podjęta ze względu na złożoność procesu zarządzania zasobami w chmurze. W obecnej formie stanowi on abstrakcję nad tym procesem, co pozwala na uzyskanie mniej skomplikowanego kodu w reszcie systemu. Istotnym celem jaki realizuje *Orkiestrator* jest jak najlepsze wykorzystanie ważnej zalety publicznych dostawców chmurowych, niedostępnej w rozwiązańach On-Premises⁶ - możliwości zrezygnowania z danej usługi w każdym momencie, co prowadzi do ograniczenia kosztów.Więcej o schemacie dostarczania serwisów można przeczytać w sekcji 3.7.6. Dodatkowo w *Orkiestratorze* zaimplementowane zostały mechanizmy niezawodności, które w bardziej szczegółowy sposób opisano w sekcji 3.7.7.

3.1.3. Silnik MDA

Tworzona aplikacja ma za zadanie udostępnić nauczycielowi prostą formę tworzenia kursu na podstawie gotowych modułów, bez znajomości szczegółów technicznych. Jest to główna motywacja użycia podejścia MDA w implementacji systemu, ponieważ pozwala ono na tworzenie bardzo wysokopoziomowych wymagań, przez nauczyciela. *Silnik MDA* będący implementacją tego podejścia, posiada rozwiązania pozwalające na automatyczny wybór optymalnych maszyn wirtualnych, używanych do przeprowadzania zajęć. Zwalnia to nauczyciela z potrzeby znajomości wymagań modułów, jak i wyboru maszyny wirtualnej, a pozwala skupić mu się tylko i wyłącznie na przygotowaniu i przeprowadzeniu w późniejszym terminie zajęć.

3.1.4. Serwis Chmurowy

Z uwagi na wymagania wysokiej dostępności serwisów, rozbudowanych mechanizmów monitorowania użycia zasobów, bezpieczeństwa i niezawodności przechowywania plików, oraz przede wszystkim z uwagi na wymaganie dostarczenia powyższych funkcjonalności w placówkach o ograniczonych zasobach zdecydowano się na wybór jednego z dostawców chmury publicznej. Serwis chmurowy musi udostępniać funkcjonalności operacji na maszynach wirtualnych oraz plikach użytkowników.

3.1.5. Baza Danych

Każda nadająca się do realnego zastosowania dynamiczna aplikacja internetowa wymaga jakieś formy trwałego przechowywania danych. Z uwagi na liczbę i stopień rozbudowania wielu wykorzystywanych w aplikacji zasobów, takich jak kursy czy serwisy, z których każdy musi zostać utrwalony w pamięci nieulotnej zdecydowano się na wykorzystanie relacyjnej bazy danych. Wydzielono dwie, logicznie niezależne bazy danych, jedna z nich wykorzystywana jest przez *Serwer*, druga przez *Orkiestrator*. Schematy baz omówiono w sekcji 3.3.

3.1.6. Aplikacja Webowa

Funkcjonalności *Aplikacji Webowej* zostały wystarczająco dokładnie opisane w sekcji 2.5. Podsystem będzie wchodził w bezpośrednią interakcję z *Serwerem* i na podstawie dostarczonych przez niego danych umożliwi korzystanie z systemu końcowym użytkownikom.

⁶On-Premises - sposób wdrażania oprogramowania cechujący się instalacją na własnym sprzęcie znajdującym się na terenie danej firmy lub organizacji

3.2. Stos technologiczny

W sekcji zostanie opisany dokładny stos technologiczny, wykorzystany do implementacji każdego z komponentów systemu. Wybory zostaną uzasadnione i w większości przypadków porównane z alternatywnymi opcjami.

3.2.1. Aplikacja Webowa

W celu zapewniania dynamiki i nowoczesności strony internetowej zdecydowano się na wykorzystanie architektury SPA (Single Page Application) opartej na technologiach związanych z językiem JavaScript. W przeciwieństwie do standardowych technologii, w których cała zawartość strony jest zwracana przez serwer jako odpowiedź na żądania wysłane przez przeglądarkę (czyli gdy użytkownik otworzył stronę internetową) architektura SPA w najprostszym przypadku polega na wysłaniu niewielkiego pliku HTML wraz z paczką skryptowego kodu w języku JavaScript umożliwiającego wygenerowanie właściwej zawartości strony. W tym celu JavaScript musi zwykle wykonać pewne zapytania HTTP do serwera, pozwala na to zbiór technologii AJAX⁷.

Główną zaletą stron internetowych opartych na tej architekturze jest ich wydajność. Jedynym co serwer musi dostarczyć jest paczka statycznych plików HTML oraz JavaScript. Nie jest on odpowiedzialny za wykonywanie zaawansowanych obliczeń czy też używanie komunikacji sieciowej z innymi komponentami umożliwiającymi wygenerowanie początkowej zawartości strony. Wszystkie te rzeczy zostają oddelegowane do klienta (czyli do przeglądarki użytkownika). Utworzone w ten sposób strony ładują się szybciej i dostarczają lepszych wrażeń z korzystania użytkownikom.

Niestety rozwiązanie to może mieć również wady, główną z nich jest bardzo wyraźnie ograniczenie jakości SEO (Search Engine Optimization), czyli indeksowania przez silniki wyszukiwania takie jak Google Search stron internetowych. W skrócie indeksowanie polega na pobieraniu przez web crawlery⁸ pliku HTML strony i przetworzeniu metadanych oraz tekstu w nim zawartego umożliwiając późniejsze wyszukanie strony na jego podstawie. Stosuje się wiele podejść pozwalających na rozwiązanie tego problemu. Najpopularniejszym jest SSR (Server Side Rendering). Polega on na wygenerowaniu jedynie początkowej strony przez serwer, zawierającej wystarczająco dużo informacji dla web crawlerów oraz wysłaniu do klienta skryptów w języku JavaScript pozwalających na jej pełne wyrenderowanie i udynamicznenie. Oczywistym problemem jest możliwość przeciążenia serwera, gdyż będzie w takim przypadku odpowiedzialny za czynności, które w przypadku architektury SPA byłyby oddelegowane do klienta.

Biorąc jednak pod uwagę środowisko w jakim produkt ma być używany problem SEO nie ma większego znaczenia. Zarówno nauczyciele jak i studenci będą posiadali linki do serwisu. Mogą one zostać dostarczone przez administratora placówki, przykładowo drogą mailową. System będzie ograniczać dostęp jedynie dla upoważnionych użytkowników, nie powinna więc wystąpić konieczność używania silnika wyszukiwania w celu odnalezienia strony.

Do implementacji systemu opartego na wyżej opisanej architekturze wykorzystano:

- **Język TypeScript** - rozszerzenie języka JavaScript umożliwiające statyczne typowanie. Kod zaimplementowany w tym języku jest kompilowany do JavaScript'u umożliwiając

⁷Asynchronous JavaScript and XML - zbiór technik pozwalających na dynamiczne, asynchroniczne generowanie treści na stronach internetowych, korzystając przy tym z komunikacji sieciowej

⁸web crawler - bot przeszukujący sieć i przetwarzający zawartość stron internetowych

tym samym uruchomienie go w przeglądarce internetowej. Wybór został podjęty na podstawie oczekiwanej stopnia rozbudowania aplikacji. Statyczne typowanie w bardzo dużym stopniu eliminuje potencjalne błędy programisty związane z nieprawidłowym założeniem co do typu zmiennych. Dodatkowo środowiska IDE (np. Visual Studio Code) służące do wytwarzania oprogramowania w tym języku dostarczają wiele narzędzi takich jak autosugestie, które istotnie zwiększą efektywność pracy.

- **React** - najpopularniejsze środowisko programistyczne do tworzenia dynamicznych aplikacji internetowych. Wybór został głównie uzasadniony ogromną ilością kompatybilnych z nim bibliotek zwiększających wydajność aplikacji oraz upraszczających ich rozwój. Dodatkowo w stosunku do konkurencji (np. Angulara) jest on istotnie szybszy, co pozytywnie wpływa na wygodę użytkowników. Ponadto na jego podstawie powstały platformy umożliwiające zniwelowanie problemu SEO poprzez stosowanie między innymi wyżej opisanego mechanizmu SSR, np. Next.js. Gdyby z pewnych nieprzewidzianych początkowo przyczyn okazało się to ważnym czynnikiem, migracja projektu napisanego w React na projekt napisany w Next.js powina być mało pracochłonna, gdyż są one w wysokim stopniu kompatybilne.
- **MUI** - zestaw bibliotek będący kompatybilną z React implementacją popularnego standardu Material Design zaprojektowanego przez Google. Służy do stylizacji komponentów na stronie, dostarczając gotowe bazowe style dla elementów takich jak przyciski czy formularze. Dodatkowo pozwala na znaczną ingerencję w bazowe style, dzięki czemu aplikacja wciąż może mieć swój unikalny wygląd. Porównując z konkurencyjnymi platformami typu Tailwind.css, MUI istotnie zwiększa wydajność programisty zachowując przy tym wystarczająco elegancki styl strony.
- **Redux** - biblioteka pozwalająca na bezpieczne i funkcyjne zarządzanie globalnym stanem aplikacji stworzonych w środowisku React. Komponenty korzystając z predefiniowanych zasad interakcji z globalnym stanem mogą go zmieniać oraz odczytywać. Jest to standard bardzo często chodzący w parze z każdą aplikacją napisaną w React.
- **React Query** - nowoczesna biblioteka umożliwiająca asynchroniczne wysyłanie zapytań HTTP. Zapewnia niezbędne mechanizmy takie jak powtarzanie zapytań w przypadku błędów związanych z komunikacją sieciową, dodatkowo zwiększa wydajność poprzez cache'owanie odpowiedzi do uprzednio wysłanych zapytań. Dostarcza funkcji aktualizowania pamięci cache, dzięki czemu zawartość strony może zostać automatycznie odświeżana.
- **Open Api Generator** - biblioteka umożliwiająca na podstawie dostarczonego schematu API (endpointy HTTP, zwarcane przez nie typy i oczekiwane typy zawartości zapytań) wygenerować stuby⁹ w języku TypeScript umożliwiające wykonywanie tych zapytań do serwera. Dzięki wykorzystaniu generatora po zdefiniowaniu API po stronie serwera nie było potrzeby ręcznego powtarzania tej pracy po stronie aplikacji webowej, implementując np. zwarcane typy obiektów.
- **I18next** - biblioteka umożliwiająca tłumaczenie na różne języki zawartości stron internetowych.

⁹stub - kod odpowiedzialny za komunikację w systemie rozproszonym

3.2.2. Back-end

Obsłuszenie zapytań HTTP wysyłanych przez aplikację webową wymaga serwera. Do komunikacji między *Serwerem* a klientem zdecydowano się na architekturę REST (REpresentational State Transfer). Architektura zakłada reprezentację obiektów, których do działania wymaga klient, jako abstrakcyjnych zasobów. Zasoby powinny być identyfikowalne za pomocą adresów URL. Serwer ma za zadanie umożliwić wykonywanie operacji na tych zasobach poprzez jasno określony interfejs programistyczny (REST API). Standard określa kilka typowych metod interakcji z zasobami, wśród nich możemy wyróżnić między innymi:

- **GET** - metoda służąca do odczytu stanu adresowanych zasobów,
- **POST** - metoda służąca do utworzenia zasobu,
- **PUT** - metoda zwykle służąca do aktualizacji zasobu,
- **DELETE** - metoda pozwalająca na trwałe usunięcie zasobu.

Metody te umożliwiają spełnienie funkcjonalności CRUD. Technologie pozwalające na implementacje serwerów zgodnie z tym standardem zostały szeroko rozwinięte w wielu językach programowania, bazując one na protokole HTTP. Jedną z rozważanych alternatyw był standard GraphQL, również oparty na protokole HTTP, jednak w tym przypadku zamiast rozróżniania operacji na podstawie opisanych wyżej metod stosuje się zapytania zwane kwerendami (ang. queries) oraz mutacjami (ang. mutations). Pierwsze z nich pozwalają na odczyt zasobu, główną różnicą między tym, a metodą GET w podejściu REST jest pozwolenie klientowi na dokładne określenie jakiego zasobu bądź kombinacji zasobów potrzebuje, bez sztywnego adresowania za pomocą URL. Drugie z nich pozwala na zmienianie stanu zasobów, w tym przypadku również klient, a nie serwer dyktuje co i jak powinno zostać zmienione. Ze względu na jeszcze względną młodość tego podejścia oraz ryzyko braku wsparcia niektórych funkcji wybrano ostatecznie podejście REST.

W sekcji 3.1 uwzględniony został logiczny podział tego podsystemu na komponenty. Zdecydowano żeby *Serwer* oraz *Silnik MDA* pracowały wspólnie w ramach jednego procesu systemu operacyjnego. *Orkiestrator* został wydzielony do osobnego, niezależnego projektu. Decyzja o fizycznej separacji tego systemu została podjęta na podstawie jego rozbudowanego i wymagającego obliczeniowo zakresu obowiązków. Podział ten umożliwia niezależną replikację i równoważenie obciążenia *Serwera* oraz *Orkiestratora*. Dodatkowo separacja daje możliwość wykorzystania (nawet jednocześnie) różnych implementacji *Orkiestratora*, co mogłoby okazać się przydatne gdyby wynikła potrzeba zmiany dostawcy usług chmurowych. W takim przypadku nie byłibyśmy zmuszeni do wprowadzania jakichkolwiek zmian po stronie *Serwera*. Ważna była by tylko implementacja tego samego zdefiniowanego API przez nowy orkiestrator. *Silnik MDA* został zintegrowany z *Serwerem*, ponieważ wymaga on struktur danych używanych oraz aktualizowanych przez *Serwer*. Szczegółowy opis tych struktur przedstawiono w sekcjach 3.3 oraz 3.4. Udostępnienie struktur przez sieć mogłoby w zauważalnym stopniu ograniczyć wydajność systemu, a ponadto wiązałoby się z istotnie zwiększonym nakładem pracy implementacyjnej.

Do komunikacji między *Serwerem*, a *Orkiestratorem* wybrano również architekturę REST. Głównym powodem była możliwość wykorzystania tych samych bibliotek, które zostały wykorzystane do komunikacji z *Aplikacją Webową*, ponadto udało się zaprojektować API *Orkiestratora* jako relatywnie proste i początkowo niewymagające bardziej zaawansowanych technologii. Jako alternatywne rozwiązanie rozważano protokół AMQP (Advanced Message Queuing Protocol), który umożliwia asynchroniczne wysyłanie wiadomości między procesami przez sieć. Wiadomości są wysyłane do specjalnego węzła zwanego brokerem i rozsyłane przez niego do

właściwych adresatów. Implementacje protokołu, przykładowo RabbitMQ, dostarczają wielu przydatnych mechanizmów QoS (Quality of Service) takich jak potwierdzenia przetworzenia wiadomości, ich priorytetyzacja czy też kolejkowanie, gdy system jest przeciążony. To rozwiązanie ostatecznie mogłoby okazać się lepsze, jednak wymagałoby początkowo znacznie większego nakładu pracy. Komponenty *Serwera* oraz *Orkiestratora* odpowiedzialne za komunikację zostały więc zaprojektowane w sposób abstrakcyjny i niezależny od samej warstwy komunikacyjnej, dzięki czemu możliwa jest jej relatywnie prosta wymiana, gdyby okazało się to konieczne.

Komunikacja między systemami wymaga uwierzytelnienia, w przypadku *Aplikacji Webowej* uwierzytelnić musi się każdy użytkownik, w przypadku *Orkiestratora* cały podsystem. Do uwierzytelnienia użytkowników wybrano technologię JWT (JSON Web Token) polegającą na przesyłaniu przez klienta w nagłówku HTTP specjalnego tokenu wygenerowanego i podpisaneego cyfrowo przez serwer po podaniu przez użytkownika loginu i hasła. Serwer po otrzymaniu żądania od klienta weryfikuje podpis i odczytuje identyfikatora oraz role użytkownika, na których podstanie przypnaje, bądź odmawia dostępu do zasobu. Jest to metoda bezstanowa, w przeciwieństwie do kiedyś dominującej metody uwierzytelniania za pomocą ciasteczek (ang. cookies) podatnej na ataki CSRF (Cross Site Request Forgery) oraz mniej wydajnej obliczeniowo z uwagi na stanowość. Do uwierzytelnienia *Orkiestratora* użyto współdzielonego klucza tajnego, wysyłanego przez *Orkiestrator* w nagłówku HTTP. Kwestie bezpieczeństwa omówiono w sekcjach 3.7.1 oraz 3.7.3.

Zgodnie z sekcją 3.1.5 jako warstwę persystencji wybrano relacyjną bazę danych. W celu zwiększenia niezależności między *Serwerem*, a *Orkiestratorem* stosowane są ostatecznie dwie bazy danych. *Serwer* nie wykorzystuje bezpośrednio żadnych przechowywanych w pamięci trwałej struktur danych używanych przez *Orkiestartor* i odwrotnie. Wybór relacyjnej bazy po stronie *Serwera* był konieczny z uwagi na rozbudowane relacje między zasobami takimi jak kursy, aktywności edukacyjne i moduły, co zostało omówione dokładniej w sekcji 3.3. Po stronie *Orkiestratora* relacje są znacznie mniej skomplikowane i można było użyć innej, bardziej przystosowanej do środowisk rozproszonych, bazy NoSQL takiej jak Cassandra. W teorii CAP (Consistency, Availability, Partition tolerance)¹⁰ bazy relacyjne spełniają warunki CA, natomiast Cassandra spełnia AP. Biorąc pod uwagę architekturę *Orkiestratora* bardziej pożądane były ostatecznie cechy AP. Z tego powodu również ta warstwa została zaimplementowana w sposób abstrakcyjny i umożliwiający ewentualną podmianę, a z uwagi na lepszą znajomość technologii zdecydowano się na użycie bazy relacyjnej.

Projektując cały system wyciągnięto wnioski, że najlepszym podejściem będzie podejście obiektowe, ponieważ, jak wyżej wspomniano, wiele podsystemów musi operować na odpowiedniej warstwie abstrakcji, którą w wygodny sposób zaimplementować pozwalają języki obiektowe. Z uwagi na chęć ponownego użycia niektórych klas i funkcji, zarówno *Serwer* (i tym samym *Silnik MDA*) jak i *Orkiestrator* zostały zaimplementowane w tym samym języku.

Do implementacji systemu opartego na wyżej opisanej architekturze wykorzystano:

- **Język Java** - obiektowy, statycznie typowany język działający na maszynie wirtualnej Javy (JVM), co pozwala na uruchomienie projektu w nim zaimplementowanego na dowolnej platformie sprzętowej, na której działa JVM. Na wybór samej Javy spośród innych znanych nam języków obiektowych takich jak Python czy C++ przyczyniło się wymaga-

¹⁰twierdzenie CAP dowodzi, że w systemie podatnym na awarie związane z komunikacją nie jest możliwe jednocześnie zapewnienie spójności danych (C), wysokiej dostępności (A) oraz odporności na podział w sieci (P)

nie statycznej typizacji, wydajność (obu tych cech nie posiada język Python) oraz potrzeba, związana z ograniczonym czasem dostępnym na implementację projektu, posiadania automatycznych mechanizmów zarządzania pamięcią (Garbage Collector, którego z kolei nie posiada C++). Ostatecznym czynnikiem była ogromna liczba kompatybilnych bibliotek umożliwiających wydajną i odporną na błędy implementację wielu potrzebnych funkcjonalności.

- **PostgreSQL** - jedna z najpopularniejszych baz relacyjnych, posiadająca szerokie wsparcie w wielu bibliotekach. Na popularność przyczyniła się otwarta implementacja (Open Source), wydajność oraz niezawodność. Sama technologia rozwijana jest od wielu lat.
- **Spring** - najpopularniejsze środowisko programistyczne służące do tworzenia serwerów webowych w technologiach opartych na JVM. W szczególności użyto platformę Spring Boot, zawierającą zbiór najczęściej potrzebnych bibliotek i konfiguracji umożliwiających wydajne wytwarzanie oprogramowania w środowisku Spring. Jako serwer aplikacyjny¹¹ wykorzystano Apache Tomcat, będący sprawdzonym, wydajnym i od wielu lat rozwijanym oprogramowaniem. Jedną z ważniejszych dla nas funkcjonalności platformy jest dostępność kontenera IoC (Inversion of Control) pozwalającego na wstrzykiwanie zależności do obiektów (Dependency Injection) i łatwe rozbudowanie systemu do bardzo dużych rozmiarów.
- **Hibernate** - technologia ORM (Object Relational Mapper), pozwalająca na mapowanie encji zapisanych w relacyjnej bazie danych na obiekty działające w JVM. Pozwala na operowanie na relacjach bazo-danowych w wygodny, obiektowy sposób, przykładowo na relacji jeden do wielu można operować jak na zwykłej tablicy. Użycie tej technologii w znacznym stopniu redukuje nakład pracy związany z obsługą niskopoziomowych zapytań SQL, transakcji itp. Technologia jest najpopularniejszą implementacją standardu JPA (Java Persistence API) obejmującego wszelkie kwestie współpracy Javy z bazą danych.
- **Java HTTP client** - nowoczesna biblioteka pozwalająca na wysyłanie zapytań HTTP, poprzez udostępnienie API do nieblokującej komunikacji asynchronicznej. Została wykorzystana do komunikacji między *Serwerem*, a *Orkiestratorem*. Dane podlegają serializacji do formatu JSON.
- **Jackson** - biblioteka służąca do serializacji i deserializacji obiektów Javy do formatu JSON i odwrotnie. Dzięki niej możliwe jest wygodne operowanie na tych samych danych w JavaScript po stronie klienta oraz w języku Java po stronie back-endu.
- **Mapstruct** - biblioteka służąca do mapowania obiektów na typy posiadające inne pola. Umożliwia efektywne tworzenie na poziomie obiektowym widoków bazo-danowych i udostępnianie różnym użytkownikom (przykładowo w zależności od uprawnień) informacji zapisanych w bazie danych w tym samym rekordzie.
- **Swagger** - biblioteka umożliwiająca generowanie specyfikacji Open API używając annotations w języku Java, specyfikacja, jak wspomniano we wcześniejszej sekcji, jest potrzebna do generowania stubów w języku TypeScript. Dodatkowo biblioteka umożliwia wygodne testowanie oraz dokumentowanie REST API z poziomu przeglądarki.
- **JWT, BouncyCastle** - biblioteki udostępniające prymitywy kryptograficzne umożliwiające zrealizowanie uwierzytelniania na podstawie JWT oraz obliczania kryptograficznej funkcji skrótu z haseł użytkowników.

¹¹ Application Server - oprogramowanie odpowiedzialne za niskopoziomową obsługę protokołu HTTP

- **Biblioteki wspierające wykorzystanie serwisu chmurowego** - zostaną dokładniej omówione w kolejnej sekcji.

3.2.3. Serwis Chmurowy

Zgodnie z założeniami naszego projektu należało podjąć decyzję, który dostawca chmurowy zapewni tworzonej aplikacji dostęp do niezbędnych zasobów. Obecnie na rynku istnieje wiele rozwiązań dostarczających usługi chmurowe. Dostępne oferty różnią się głównie zróżnicowaniem dostępnych usług, pulą zasobów obliczeniowych, niezawodnością i poziomem wsparcia dla programowego wykorzystywania interfejsów do interakcji z systemami dostawcy chmurowego.

Pierwszy istotny podział chmur można przeprowadzić ze względu na grupę docelową, do której ma ta usługa trafić. Wyróżniamy:

- **Chmury publiczne** - Oferowane dla szerokiego grona odbiorców, najczęściej cechują się dużą liczbą oferowanych usług, praktycznie nieogranicznymi zasobami obliczeniowymi, wysokimi standardami bezpieczeństwa i wsparciem dla deweloperów w postaci dokumentacji oferowanych serwisów. Z tego typu chmur korzystają małe i duże przedsiębiorstwa, przez co, w razie braków w dokumentacji, najczęściej można znaleźć rozwiązanie napotkanych problemów dzięki pomocy społeczności korzystającej z tych usług na co dzień. Dodatkowo najczęściej oprócz zwykłego interfejsu HTTP oferowane są biblioteki do popularnych języków programowania, co ułatwia proces wytwarzania oprogramowania. Do takich chmur możemy zaliczyć Microsoft Azure, Google Cloud i AWS.
- **Chmury akademickie** - Jest to rozwiązanie oferowane przez uczelnie lub organizacje oświatowe w celach edukacyjnych. W porównaniu z chmurami publicznymi nie możemy liczyć na równie rozbudowany wachlarz oferowanych usług, wsparcie społeczności, czy oferowanie bibliotek do interakcji z serwisem z wybranego przez nas języka programowania. Przykładem takich chmur są Małopolska Chmura Edukacyjna i Chmura Uniwersyteckiego Centrum Obliczeniowego Uniwersytetu w Białymostku.
- **Chmury prywatne** - Z tego typu chmury korzystają najczęściej duże korporacje potrzebujące obsłużyć bardzo wysoki ruch do swoich serwisów. Zaletą tego rozwiązania jest możliwość dostosowania architektury systemu do swoich potrzeb oraz potencjalne oszczędności, gdy zużycie zasobów jest na tyle duże, że koszta związane z ich wykorzystaniem u publicznego dostawcy przekraczałyby możliwe oszczędności związane z brakiem nadmiarowości w posiadanych systemach. Wybór takiego rozwiązania wiąże się z koniecznością utrzymywania i serwisowania zasobów sprzętowych, co z kolei wymaga wyspecjalizowanej kadry, poświęconej tylko temu zadaniu.

Ostatecznie, chmura publiczna została wybrana ze względu na zdecydowaną przewagę w stosunku do pozostałych opcji w przypadku zastosowania jej w niewielkim projekcie.

Spośród dostępnych na rynku dostawców usług chmurowych, pod uwagę zostali wzięci trzej najbardziej rozwinięci i rozpoznawalni: Microsoft, Google i Amazon. Oferty wyżej wymienionych dostawców są do siebie wysoce podobne i ciężko znaleźć funkcjonalność która występuje tylko na jednej platformie. W fazie planowania projektu sporządzone zostało zestawienie najważniejszych aspektów rozwiązań oferowanych przez wyżej wymienionych dostawców.

Cecha	Google Cloud	Amazon Web Services	Microsoft Azure
Darmowa wersja próbna	Tak, 90 dni, \$300 do wykorzystania na usługi	Tak, 1 rok, ograniczony dostęp do niektórych usług	Tak, 1 rok, 200\$ na płatne usługi przez pierwsze 30 dni
Usługa maszyn wirtualnych	Compute Engine	EC2	AVM
Przechowywanie plików	Cloud Storage	S3	Azure Blob
Bazy danych	Big Query - SQL, FireStore - NoSQL	Amazon RDS - SQL, Amazon DynamoDB - NoSQL	MySQL - SQL, CosmosDB - NoSQL
Repozytorium obrazów dockerowych	Container Registry	Amazon Elastic Container Registry	Container Registry
Runner Dockera	Cloud Run	Brak	Brak

Tabela 3: Podsumowanie funkcjonalności platform chmurowych

Analizując podsumowanie ustalono, że wybierając którykolwiek z dostawców założenia projektu zostaną spełnione, lecz finalny wybór padł na Google Cloud. Decyzja została podjęta na podstawie kilku, nie tylko technicznych, czynników:

- najlepsze warunki do testowania swoich usług, względnie długi czas 3 miesiący bez limitów dotyczących typu wykorzystywanych serwisów, ograniczony jedynie budżetem,
- doświadczenie z usługami oferowanymi przez Google Cloud wśród dwóch z czterech członków zespołu,
- możliwość wykorzystania Docker runnera, pozwalającego spełnić początkowe założenia projektu.

W miarę rozwoju projektu po stronie *Orkiestratora* okazało się, że wykorzystanie Cloud Run nie będzie najlepszym wyjściem. Jest to względnie nowa usługa i w momencie implementacji *Orkiestratora* biblioteka do interakcji z nią dopiero powstawała. To spowodowało, że zapadła decyzja o wykorzystaniu maszyn wirtualnych do realizacji funkcjonalności przygotowywania serwisów.

Technologie i serwisy, które finalnie wykorzystano w projekcie:

- **Compute Engine** - serwis pozwalający na uruchamianie zróżnicowanej gamy predefiniowanych maszyn wirtualnych, oferujący również możliwość dostosowywania poszczególnych parametrów pod bardziej indywidualne potrzeby. Pozwala na tworzenie maszyn wirtualnych od podstaw, a także z przygotowanych wcześniej obrazów dyskowych lub obrazów całych maszyn. Koszty w jego przypadku naliczają się za czas życia maszyny z możliwością uzyskania zniżek np. za równomierne i długotrwałe zużycie zasobów. W projekcie wykorzystujemy je jako izolowane środowiska dla studentów, w obrębie których mogą realizować aktywności edukacyjne.

- **Cloud Storage** - system przechowujący dane jako obiekty (ang. object storage) w izolowanych przestrzeniach zwanych kubelkami (ang. buckets), odpowiednik popularnego Amazon S3. W przeciwieństwie do tradycyjnych systemów plików oferujących hierarchiczną strukturę danych i możliwość częstej modyfikacji jego elementów, Cloud Storage stawia na niezawodność, wersjonowanie plików i replikację danych w wielu strefach dostępności. Dobrze sprawdza się jako repozytorium statycznych zasobów dla stron internetowych, a w przypadku naszego projektu znalazł zastosowanie jako repozytorium plików utworzonych przez studentów w trakcie aktywności edukacyjnych.
- **Biblioteki Google Cloud** - użytkownicy Google Cloud mogą skorzystać z zestawu bibliotek¹², aby w łatwiejszy sposób podjąć interakcję z usługami chmurowymi. Eliminuje to wiele problemów, które mogłyby zaistnieć, próbując wykorzystać sam interfejs HTTP. W całym projekcie wykorzystane zostały dwie biblioteki: google-cloud-storage oraz google-cloud-compute.
- **Protokół SSH** - jest to protokół zapewniający bezpieczne, szyfrowane połączenie do zdalnych instancji systemów komputerowych. Znalazł on zastosowanie zarówno bezpośrednio, jak i pośrednio w podsystemie *Orkiestratora*. W pierwszej fazie wdrażania oprogramowania *Orkiestrator* próbuje maszyny wirtualne próbując nawiązać połączenie za pomocą tego protokołu, żeby sprawdzić, czy dana instancja jest już gotowa do podjęcia z nią interakcji. Następnie SSH jest wykorzystywane przez Ansible do przygotowania odpowiednich serwisów.
- **Ansible** - jest to zestaw narzędzi pozwalający na zarządzanie konfiguracją i wdrażaniem aplikacji w systemach komputerowych wymagający od zdalnych instancji jedynie obsługi protokołu SSH. Posiada własny deklaratywny język, w formacie YAML, korzystający z tak zwanych modułów, dzięki czemu opis stanu maszyny wirtualnej pozbawiony jest imperatywnego przepływu sterowania na rzecz bardziej wysokopoziomowego opisu zadań do wykonania na maszynie. Wiele dostępnych w Ansible modułów oferuje własność idempotentności¹³, co pozwala na dużo prostszą obsługę ponawiania w razie niepowodzeń. Dzięki tym cechom Ansible jest dużo prostsze w wykorzystaniu od tradycyjnych skryptów bashowych zastosowanych w pracy *On-demand provisioning of educational cloud-based services* [2].

¹²Lista dostępnych bibliotek GCP <https://mvnrepository.com/artifact/com.google.cloud/google-cloud-compute-bom>

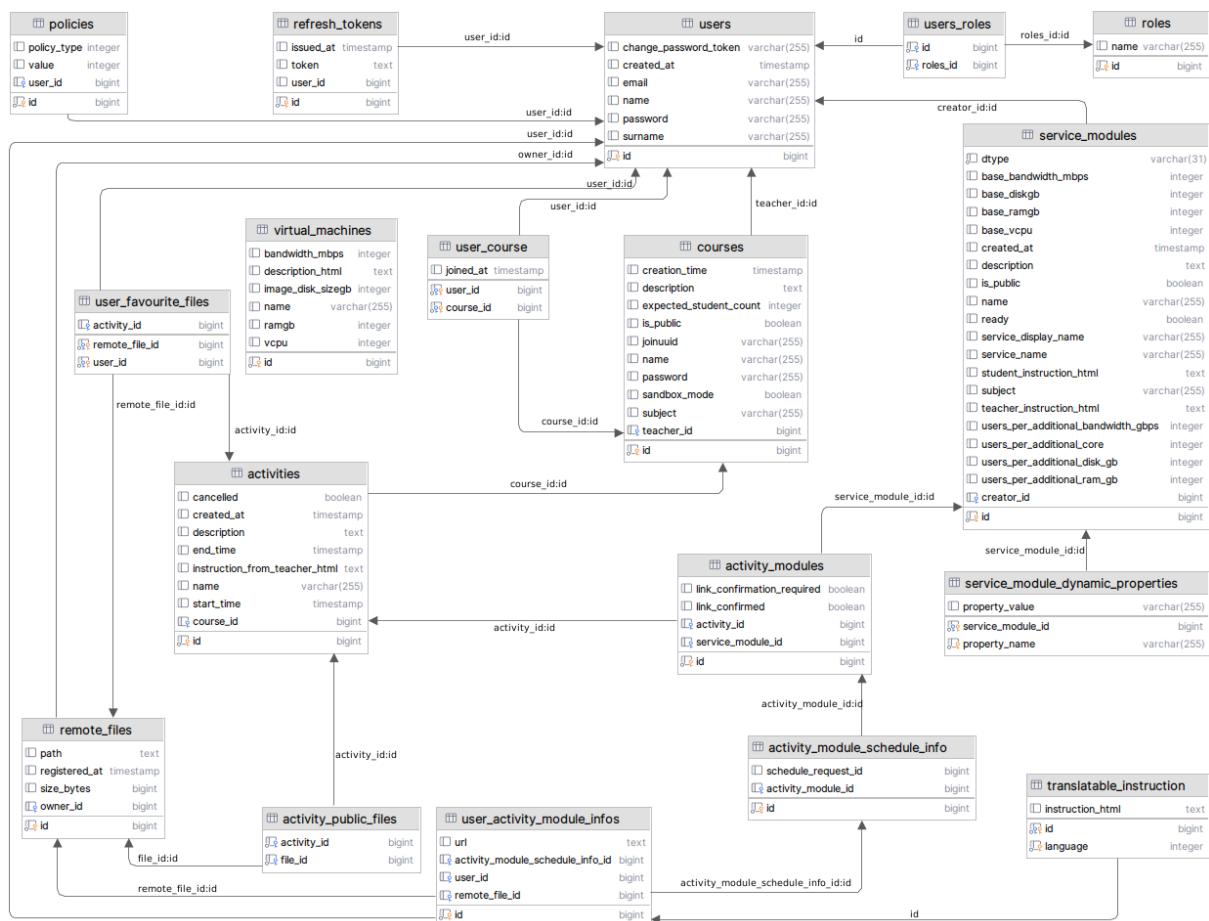
¹³Idempotentność - własność, wywodząca się z matematyki, stanowiąca o tym, że niektóre operacje można wykonać wielokrotnie, bez zmiany wyniku otrzymanego z ich pierwszego wykonania

3.3. Model danych

W sekcji zostanie przedstawiony model danych, który stanowi podstawę całego systemu. Zostaną zaprezentowane schematy baz danych *Serwera* oraz *Orkiestratora* wraz z opisem ich najważniejszych elementów. W kolejnej sekcji zostanie wy tłumaczone jak architektura MDA współgra z tym modelem w celu rozwiązania problemu organizacji zajęć online.

3.3.1. Schemat bazy danych *Serwera*

Rysunek 11 przedstawia schemat bazy danych *Serwera*, w górnej części umieszczone zostały tabele związane z użytkownikami, w centralnej części znajdują się tabele związane z kursami, a w prawej części z modułami lekcyjnymi.



Rysunek 11: Schemat bazy danych *Serwera*

W dalszej części przybliżone zostanie przeznaczenie najważniejszych tabel oraz krótko opisane zostaną ich atrybuty:

- **Użytkownicy (Users)** - tabela reprezentuje użytkowników w systemie, poza typowymi atrybutami takimi jak adres email, imię, nazwisko i hasło zapamiętane są tutaj informacje administracyjne takie jak data utworzenia konta. Tabela jest w relacji wiele do wielu z tabelą **Roles**, co oznacza, że każdy użytkownik może mieć wiele ról w systemie, role odpowiadają tym, które zostały przedstawione w sekcji 2.1 i dla przypomnienia są to: Administrator, Student, Nauczyciel i Nauczyciel techniczny. Z tabelą jest również powiązana

tabela **Refresh Tokens** związana z uwierzytelnianiem, której przeznaczenie dokładniej opisano w sekcji 3.7.1.

- **Kursy (Courses)** - tabela reprezentuje kursy w systemie. Jednym z głównych wymagań funkcjonalnych systemu była możliwość tworzenia kursów, w skład których wchodzą aktywności edukacyjne. W tabeli przechowywane są informacje podawane przez twórcę kursu, czyli nauczyciela, takie jak nazwa, opis, hasło do dołączenia do kursu, informacja czy kurs jest publiczny (czy każdy student może go wyszukać, czy tylko student, który dostał do niego link), oraz kilka pól umożliwiających zarządzanie kursem po stronie systemu, przykładowo czy dany kurs jest środowiskiem testowym które powinno zostać automatycznie usunięte. Tabela jest w relacji wiele do wielu z użytkownikami za pośrednictwem tabeli **User Course**, co pozwala na określenie jacy studenci należą do kursu i kiedy do niego dołączyli. Ponadto każdy kurs posiada dokładnie jednego twórcę - nauczyciela, w związku z tym mamy kolejną relację wiele do jednego z tabelą **Users**.
- **Aktywności edukacyjne (Activities)** - tabela reprezentuje aktywności edukacyjne. Każdy kurs może składać się z wielu aktywności utworzonych przez nauczyciela. W tabeli przechowujemy atrybuty takie jak nazwa, opis aktywności, instrukcja podana przez nauczyciela oraz data i czas trwania aktywności.
- **Moduły lekcyjne (Service Modules)** - nauczyciel do każdej aktywności może dobrać wiele modułów lekcyjnych (dla przypomnienia, przykładowym modułem jest Jupyter z gotowym notebookiem do zajęć z uczenia maszynowego). Tabela zawiera dane konfiguracyjne dostępnych w systemie serwisów, takich jak Jupyter czy QuizApp, wprowadzone przez nauczyciela technicznego, które dokładniej opisano w sekcji 3.4. W tabeli dodatkowo przechowywane są standardowe informacje takie jak nazwa, opis, instrukcje wykorzystania modułu dla nauczyciela i ucznia oraz nazwa serwisu który został wybrany do utworzenia modułu, a także informacja o twórcy modułu. Tabela jest w relacji jeden do wielu z tabelą **Service Module Dynamic Properties**, która pozwala na generyczną implementację obsługi modułów tak, aby dodanie kolejnych serwisów nie wymagało zmian w modelu danych, zostało to wyjaśnione w sekcji 3.7.8.
- **Activity Modules** oraz **User Activity Module Infos** - są to tabele łącznikowe między aktywnością, modułami lekcyjnymi w niej wykorzystanymi oraz uczestnikami kursu. **Activity Module** odpowiada wyborowi modułu lekcyjnego do danej aktywności, przechowywane są tutaj dodatkowo informacje czy dostęp do modułu może już być przyznany uczniom i czy wymagane jest potwierdzenie tej akcji przez nauczyciela (przykładowo, przy wyborze quzu nauczyciel może nie chcieć, żeby uczniowie mieli do niego dostęp zaraz po rozpoczęciu zajęć). Z każdym **Activity Module** są dodatkowo powiązani uczestnicy kursu, którzy muszą dostać dane dostępowe do tego modułu - linki oraz instrukcje dołączenia. Informacje te są przechowywane w tabeli **User Activity Module Infos**. Pozałe powiązane dane takie jak **Activity Module Schedule Info** dotyczą wewnętrznego zarządzania cyklem życia uruchomionego modułu.
- **Zdalne pliki (Remote Files)** - tabela przechowuje metadane plików zapisanych w chmurze. Lokalnie zapisujemy dane takie jak ścieżka pliku, rozmiar oraz dane właściciela. Tabela jest powiązana z **Activities**, ponieważ nauczyciel do każdej aktywności może dodać pliki (np. z instrukcjami), które uczniowie powinni pobrać. Mamy też powiązanie z **User Activity Module Infos** dzięki czemu możemy przechowywać i przypisywać do odpowiednich modułów lekcyjnych wyniki prac uczniów. Dokładniejszy opis sposobu operacji na plikach został opisany w sekcji 3.7.2.

- **Maszyny wirtualne (Virtual Machines)** oraz **Polityki (Policies)** - tabele związane z MDA, opisane w sekcji 3.4.

```

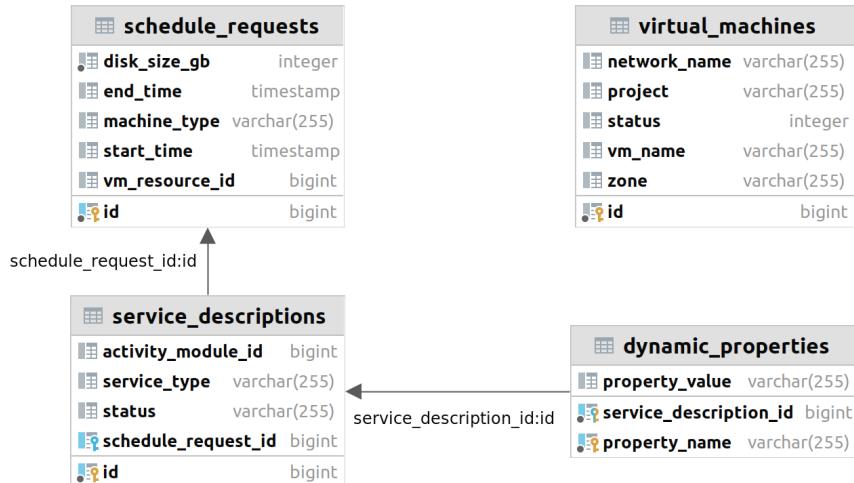
name: Programowanie w języku Python
subject: Informatyka
description: Kurs przechodzi przez podstawy programowania w języku Python.
joinUUID: 8cc7cadb-eab0-48c8-99a6-29348258d308
password: haslo
creationTime: '2022-11-20T11:24:08.817246901Z'
isPublic: false
sandboxMode: false
expectedStudentCount: 2
activities:
- name: Pętle, konstrukcje warunkowe
  description: Wprowadzenie do korzystania z języka Python.
  startTime: '2022-07-29T17:30:40Z'
  endTime: '2022-07-29T19:30:40Z'
  instructionFromTeacherHtml: <h1>Przed zajęciami należy przeczytać dokumentacje Pythona</h1>
  createdAt: '2022-11-20T11:24:08.827642492Z'
  modules: # dane modułów zostaną omówione później
students: # dane uczniów pominięto, podobne do danych nauczyciela, tylko rola = STUDENT
teacher:
  name: Włodzimierz
  surname: Biały
  email: wbialykntp@gmail.com
  password: hashowane-haslo
  roles:
  - TEACHER
  - TECHNICAL_TEACHER

```

Rysunek 12: Przykładowy kurs wraz z danymi nauczyciela

W celu lepszego zobrazowania powyższych informacji rysunek 12 przedstawia dane przykładowego kursu w formacie yaml. Przykład pokazuje kurs z jedną aktywnością edukacyjną, wykorzystującą moduły lekcyjne których dane pominięto. Nauczyciel ma jednocześnie role zwykłego oraz technicznego nauczyciela, co pozwala mu na tworzenie kursów oraz modułów lekcyjnych.

3.3.2. Schemat bazy danych *Orkiestratora*



Rysunek 13: Schemat bazy danych *Orkiestratora*

W schemacie bazy danych *Orkiestratora* znajdują się 4 tabele:

- **Żądania przygotowania serwisów (Schedule Requests)** - nadrzędna tabela dla wszystkich informacji związanych z pojedynczym żądaniem otrzymanym od *Serwera*. Zawiera takie informacje, jak specyfikacja maszyny wirtualnej i czas życia serwisów.
- **Specyfikacja serwisów (Service Descriptions)** - tabela zawiera opis pojedynczego serwisu dostarczony przez *Serwer* oraz dodatkowo status serwisu w celu śledzenia jego cyklu życia i zapewnienia niezawodności. Dla każdego żądania *Serwera* w tej tabeli może znaleźć się wiele rekordów.
- **Dynamiczne parametry (Dynamic Properties)** - miejsce zapisu dynamicznych parametrów rozumianych przez poszczególne serwisy.
- **Maszyny wirtualne (Virtual Machines)** - tabela przechowuje informacje o używanych w systemie maszynach wirtualnych z platformy Google Cloud. Warto zwrócić uwagę na brak relacji z tabelą **Schedule Requests**. Dzięki temu w razie zmiany dostawcy maszyn wirtualnych nie będzie konieczności modyfikacji kodu odpowiedzialnego za obsługę żądań od *Serwera*.

3.4. Implementacja wzorca MDA

W sekcji zostanie przedstawiona implementacja wzorca MDA w naszej aplikacji. Ukażane zostaną wszystkie trzy z wykorzystywanych modeli (CIM, PIM, PSM) uzupełnione o przykładowe dane oraz mechanizmy odpowiadające za ich translację wraz z uwzględnieniem polityk oraz repozytorium dostępnych maszyn wirtualnych.

3.4.1. Modele danych

W podejściu MDA wykorzystano następujące modele danych:

a) CIM - Computation Independent Model

Pierwszy model danych wykorzystywany w przetwarzaniu zapytania od nauczyciela o stworzenie aktywności edukacyjnej. Przechowuje dane aktywności edukacyjnej odnośnie modułów lekcyjnych przez nią używanych. Danymi modelu są też: identyfikator nauczyciela oraz liczba uczestniczących studentów, dostęp do tych informacji jest możliwy dzięki wykorzystaniu danych zapisanych w kursie, do którego należy aktywność. Rysunek 14 przedstawia przykładowy log wypisany przez *Silnik MDA* zawierający CIM, w kursie uczestniczy dwóch studentów, a do aktywności wybrane zostały dwa moduły.

```
Cim(
    ServiceModuleIds=[1, 3],
    teacherId=2,
    studentsNumber=2
)
```

Rysunek 14: Log wygenerowanego modelu CIM, sformatowany dla lepszej czytelności

Przechowywanie w tym modelu danych identyfikatorów modułów lekcyjnych pełni kluźcową rolę w umożliwieniu translacji go na dalsze modele. To właśnie z nich *Silnik MDA* wyciąga informacje o wymaganiach sprzętowych dla danej aktywności. Moduły

lekcyjne dzielą się na dwa rodzaje: **SharedServiceModule** oraz **IsolatedServiceModule**. Pierwszego z nich można użyć w celu zaplanowania zajęć prowadzonych w formie *jedna instancja serwisu* → *wielu użytkowników*, drugi natomiast umożliwia używanie jednej instancji serwisu przez pojedynczego użytkownika. Należy zwrócić uwagę, że podczas tworzenia modułu przez nauczyciela technicznego ma on możliwość stworzenia obu tych rodzajów modułów (jeśli tylko wykorzystywany serwis na to pozwala, przykładowo izolowana wideokonferencja nie miałaby większego sensu). Ma to na celu umożliwienie korzystania z przygotowanych przez siebie pomocy na różne sposoby. Rysunki 15 i 16 przedstawiają odpowiednio przykładowy moduł izolowany i przykładowy moduł współdzielony, w obu przypadkach bazują one na serwisie Jupyter, dane techniczne zostały ustalone empirycznie.

```
IsolatedServiceModule(super=ServiceModule(
    name=Klasy w Pythonie,
    subject=INFORMATYKA,
    description="Moduł przybliża wiedze o klasach w pythonie",
    teacherInstructionHtml="Wybierz ta lekcje po lekcji o zmiennych",
    studentInstructionHtml="Przypomnij sobie wiedze z lekcji o zmiennych",
    serviceName=JUPYTER,
    isPublic = true,
    ready=true,
    createdAt=2022-05-29T21:30:40,
    dynamicProperites={notebookLocation=1},
    creatorId=2,
    baseVcpu=1,
    baseRamGB=1,
    baseDiskGB=1,
    baseBandwidthMbps=2048
)
)
```

Rysunek 15: Pierwszy z serwisów lekcyjnych wykorzystany w CIM-ie (izolowany)

Moduły lekcyjne składają się z danych technicznych i nietechnicznych:

- **Dane techniczne** - odgrywają ważną rolę przechowując wymagania, które muszą spełnić wybrane przez MDA maszyny wirtualne w celu przeprowadzenia zajęć. Na wymagania składają się: baseVcpu - podstawowa liczba wirtualnych procesorów, baseRamGB - podstawowa liczba gigabajtów pamięci Ram, baseDiskGB - podstawowa liczba gigabajtów pamięci dyskowej, baseBandwidthMbps - podstawowa przepustowość łącza.
- **Dane nietechniczne** - reszta danych, która nie ma wpływu na działanie *Silnika MDA*, przechowuje informacje dotyczące instrukcji do zajęć, czy użytego do realizacji ich serwisu.

W przypadku użycia modułu współdzielonego zwrócono uwagę na problem z podstawowymi wymaganiami modułu. Korzystanie przez wielu użytkowników z serwisu na

jednej maszynie wirtualnej, może spowodować, zwiększenie się zapotrzebowania na dostarczane przez nią zasoby, takie jak przepustowość łącza, czy ilość pamięci RAM. Aby zapobiec problemom z działaniem serwisów utworzono drugi rodzaj modułów lekcyjnych o nazwie **SharedServiceModule**. Posiada on dane pozwalające zwiększać wymienione wcześniej, bazowe wymagania modułu w zależności od liczby uczestników danej aktywności edukacyjnej.

```
SharedServiceModule(super=ServiceModule(
    name=Funkcje w Pythonie2,
    subject=INFORMATYKA,
    description="Moduł ma na celu poszerzyć wiedzę o funkcjach w Pythonie",
    teacherInstructionHtml="Wybierz ta lekcje po lekcji: Funkcje w Pythonie 1",
    studentInstructionHtml="Przypomnij sobie wiedzę z poprzednich zajęć o funkcjach",
    serviceName=JUPYTER,
    isPublic = true,
    ready=true,
    createdAt=2022-05-29T21:30:40,
    dynamicProperties={notebookLocation=1},
    creatorId=2,
    baseVcpu=1,
    baseRamGB=1,
    baseDiskGB=1,
    baseBandwidthMbps=1024
),
    usersPerAdditionalCore=15,
    usersPerAdditionalRamGb=15,
    usersPerAdditionalDiskGb=2,
    usersPerAdditionalBandwidthGbps=2
)
```

Rysunek 16: Drugi z serwisów lekcyjnych wykorzystany w CIM-ie (współdzielony)

b) PIM - Platform Independent Model

Drugi model danych, uzyskiwany na podstawie CIM-a, mający na celu opisanie wymagań sprzętowych jakie powinny spełniać dwa rodzaje maszyn wirtualnych, które będą umożliwiać dostęp do serwisów odpowiednio dla nauczyciela oraz uczniów. Rysunek 17 przedstawia przykładowy log wypisany przez *Silnik MDA* zawierający PIM powstały w wyniku niżej opisanej translacji z pokazanego na rysunku 14 modelu CIM.

Występowanie dwóch rodzajów maszyn: **teacherVm** i **studentVm** wynika z użycia w aktywności edukacyjnej dwóch rodzajów modułów lekcyjnych. Maszyna wirtualna nauczyciela jest tworzona zawsze, umożliwia ona dostęp do modułów izolowanych nauczycielowi oraz do modułów współdzielonych wszystkim uczestnikom kursu. W przypadku studentów, na ich maszynach *Orkiestrator* instancjonuje tylko serwisy w trybie izolowanym. Z takiego rozbicia wynika różnica w wymaganiach, które muszą spełniać te dwa rodzaje maszyn, jak i prawdopodobieństwo że w przypadku użycia wyłącznie modułów współdzielonych zniknie potrzeba oddzielnej instancji maszyny wirtualnej dla każdego studenta. Z tego powodu model PIM składa się z dwóch bloków informujących o wymaganiach technicznych, jakie muszą spełniać wybrane później typy maszyn. Bloki te przechowują także dane odnośnie liczby maszyn, czy listy modułów, które mają na nich działać.

```

Pim(
    teacherVm=PimVmInfo(
        amount=1,
        ServiceModuleIds=[1, 3],
        vcpu=2,
        ramGB=2,
        imageDiskSizeGB=3,
        bandwidthMbps=4096
    ),
    studentsVms=Optional[PimVmInfo(
        amount=2,
        ServiceModuleIds=[1],
        vcpu=1,
        ramGB=1,
        imageDiskSizeGB=1,
        bandwidthMbps=2048
    )]
)

```

Rysunek 17: Log modelu PIM, dane są wynikiem przetwarzania wcześniej zamieszczonego modelu CIM

Użycie modelu pośredniego w podejściu MDA, jakim jest model PIM, pozwala na odzielenie procesu analizy wymagań technicznych i sprawdzania polityk od metody wyboru maszyny wirtualnej. Umożliwia to późniejsze modyfikacje procesu wyboru maszyn wirtualnych jako osobnego bloku działania *Silnika MDA*, bez potrzeby zmian implementacji całego podejścia.

c) PSM - Platform Specific Model

Trzeci i ostatni model danych wykorzystywany w MDA, mający na celu przedstawienie konkretnych typów maszyn, na których zostaną przeprowadzone zajęcia, a których instancjonowanie zostanie następnie zlecone *Orkiestratorowi*. Rysunek 18 przedstawia przykładowy log wypisany przez *Silnik MDA* zawierający PSM powstały w wyniku niżej opisanej translacji z pokazanego na rysunku 17 modelu PIM.

PSM posiada, analogicznie jak PIM, podział na dwa rodzaje maszyn, które będą potrzebne do prowadzenia zajęć. Ten model jest efektem wyjściowym *Silnika MDA*, przechowuje on kompletne dane które pozwalają *Orkiestratorowi* na zaplanowanie i uruchomienie w odpowiednim czasie maszyn o konkretnych typach i wielkościach pamięci dyskowej. Przypisanie rodzajom maszyn modułów ma na celu poinformowanie *Orkiestratora* jakie serwisy powinny zostać uruchomione na poszczególnych maszynach wirtualnych.

```

Psm(
    teacherVm=PsmVmInfo(
        amount=1,
        ServiceModuleIds=[1, 3],
        machineType="e2-standard-4",
        disk=13
    ),
    studentsVms=Optional[PsmVmInfo(
        amount=2,
        ServiceModuleIds=[1],
        machineType="e2-medium",
        disk=11
    )]
)

```

Rysunek 18: Log modelu PSM, dane są wynikiem przetwarzania wcześniej zamieszczonego modelu PIM

3.4.2. Wykorzystywane modele bazy danych

Silnik MDA, poza opisanymi wcześniej modułami lekcyjnymi, korzysta jeszcze z dwóch modeli z bazy danych *Serwera*. Są nimi **Polityki** i **Maszyny Wirtualne**.

Polityki

Polityki występują w systemie jako narzędzie pozwalające administratorowi na ograniczanie dostępu do zasobów możliwych do wykorzystania przez nauczycieli. Na politykę składają się:

- **Typ polityki** - opisują wymaganie jakie musi być spełnione, możliwe typy polityk określa enum *PolicyType*, znajdują się tam takie wymagania jak: maksymalna liczba rdzeni lub pamięci wykorzystywanej maszyny wirtualnej, maksymalna długość aktywności edukacyjnej, czy też maksymalna liczba studentów w kursie.
- **Użytkownik** - polityka musi być przypisana do konkretnego nauczyciela.
- **Wartość** - liczba jakiej dany współczynnik nie może przekroczyć, aby polityka była spełniona.

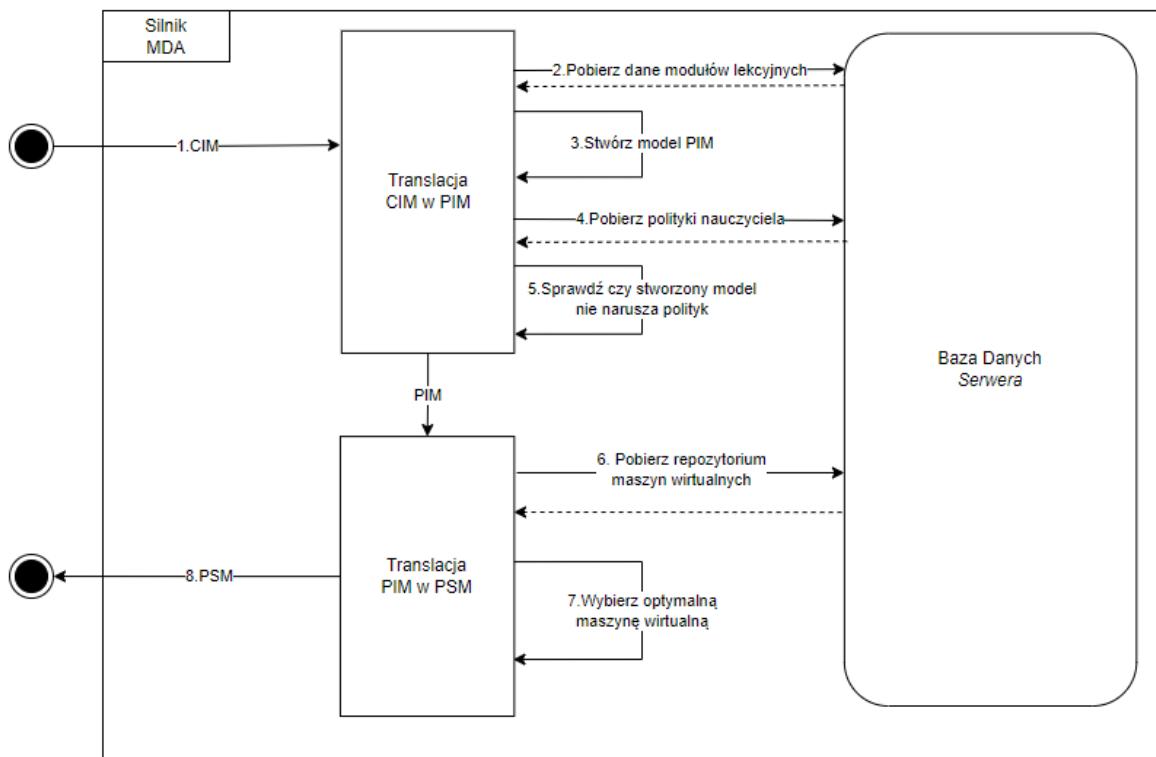
Administrator ma możliwość modyfikowania polityk nauczycieli, przy tworzeniu konta nauczyciela przypisywane są domyślne polityki.

Maszyny Wirtualne

Serwer w swojej bazie danych przechowuje takie informacje, dotyczące możliwych do wykorzystania maszyn wirtualnych, jak nazwę, udostępnianą przez maszynę liczbę rdzeni, pamięci RAM, czy przepustowość łącza, a także minimalny rozmiar pamięci dyskowej potrzebny do korzystania z konkretnych maszyn. Dzięki znajomości danych technicznych maszyn *Silnik MDA* może dobierać optymalnie rodzaj maszyn do każdej z aktywności lekcyjnych. *Silnik*, a przez co cała aplikacja, korzysta tylko z maszyn dostępnych w bazie danych, więc rolą administratora jest dodawanie, usuwanie oraz modyfikacja danych opisujących możliwe do wykorzystania maszyny. Dokładne zalecenia dla administratora opisano w sekcji 3.8.

3.4.3. Translacje modeli

Główym zadaniem *Silnika MDA* jest translacja modeli z CIM w PIM, a następnie z PIM w PSM, dla każdej planowanej aktywności w systemie. Powinien on jednocześnie zwracać uwagę na polityki przypisane nauczycielowi tworzącemu daną aktywność oraz na zbiór dostępnych maszyn wirtualnych. Efektem jego działania powinien być model PSM, który pozwala zaplanować i przygotować zajęcia za pośrednictwem *Orkiestratora*.



Rysunek 19: Diagram przepływu danych w *Silniku MDA*

Diagram 19 przedstawia działanie *Silnika MDA*. Dostaje on na wejściu model CIM, który poddaje następnie dwóm translacjom:

a) Translacja CIM -> PIM

Translacja zaczyna się od pobrania danych dotyczących modułów lekcyjnych. Na ich podstawie *Silnik* decyduje czy do przeprowadzenia danej aktywności edukacyjnej potrzebne są osobne maszyny wirtualne dla każdego z uczniów, w tym celu sprawdza występowanie modułów izolowanych. Następnie tworzy on PIM, który składa się z jednej lub dwóch rodzajów maszyn (nauczyciela i studenta). Tak wygenerowany model przed przesaniem do kolejnej translacji przechodzi przez proces sprawdzania polityk. *Silnik* pobiera z bazy danych *Serwera* polityki przypisane tworzącemu aktywność nauczycielowi i sprawdza czy są one spełnione przez każdy z dwóch rodzajów maszyn wirtualnych. W przypadku niezgodności wymagań PIM-u z politykami, proces zostaje przerwany, a *Aplikacja Webowa* informuje nauczyciela tworzącego aktywność o tym, która polityka zabrania mu stworzenia zajęć oraz proponuje mu zmianę wymagań, lub kontakt z administratorem w celu rozwiązania problemu. W przeciwnym przypadku wygenerowany model PIM zostaje zaakceptowany i przesłany do kolejnej translacji.

b) Translacja PIM -> PSM

Translacja rozpoczyna się od pobrania danych repozytorium maszyn wirtualnych oraz przefiltrowania ich, w celu znalezienia tych maszyn, które spełniają wymagania zawarte w modelu PIM. Jeśli w repozytorium maszyn *Serwera* nie istnieją maszyny spełniające wymagania to proces jest przerywany w analogiczny sposób jak w pierwszej translacji, a nauczyciel zostaje poinformowany o możliwości zmiany wymagań oraz kontaktu z administratorem. Jeśli natomiast lista typów maszyn spełniających wymagania jest dłuższa od pojedynczej instancji, następuje filtracja mająca na celu wybranie maszyny wymagającej najmniejszych zasobów, które jednak umożliwiają przeprowadzenie zajęć. Wybór najsłabszej, ale wystarczającej maszyny ma na celu minimalizację kosztów związanych z działaniem serwisów. Efektem tego działania jest końcowy model PSM, składający się z jednego lub dwóch rodzajów maszyn, które po zainstancjonowaniu przez *Orkiestrator* pozwolą na dostęp do serwisów, dzięki któremu uczestnicy kursu będą mogli wziąć udział w utworzonej aktywności edukacyjnej.

PSM będący wyjściem drugiej translacji staje się automatycznie wyjściem procesu zachodzącego w *Silniku MDA* i jest przekazywany do *Serwera*, który wykorzysta go jako parametr wysyłany do *Orkiestratora* w celu zaplanowania zajęć. Praca *Silnika MDA* kończy się na tym etapie.

3.5. Zintegrowane serwisy

Główną funkcjonalnością systemu jest umożliwienie prowadzenia zajęć wykorzystując skonfigurowane serwisy. Nauczyciel techniczny musi mieć możliwość wprowadzenia specyfikacji technicznej serwisu, żeby nauczyciel mógł go łatwo wykorzystać bez konieczności rozumienia, a nawet czytania tej specyfikacji. Ostatecznie serwis ma być dostępny dla uczniów i nauczyciela podczas zajęć za pośrednictwem przeglądarki internetowej.

Jednym z głównych celów od strony implementacyjnej było umożliwienie integracji z naszym systemem wielu różnorodnych serwisów. Szczególnie ważna była łatwość takiej integracji z punktu widzenia programistów, żeby umożliwić rozwój funkcjonalności systemu bez konieczności wprowadzania zmian w jego już zaimplementowanym rdzeniu. Wynikiem takiego podejścia są z założenia modularne i stosunkowo łatwe w utrzymaniu systemy, jako przykład (o dużo większej skali) można podać Amazon Web Services.

W celu pozwolenia użytkownikom na rzeczywiste wykorzystanie systemu oraz w celu sprawdzenia czy implementacja jest wystarczająco generyczna aby można było stwierdzić, że jest on łatwo rozwijalny zintegrowano trzy serwisy z jasno określonym zakresem zastosowań: Jupyter (usługa notebooków), Jitsi Meet (usługa wideokonferencji) oraz QuizApp (usługa quizów). Ponadto w celu umożliwienia wykorzystania innych serwisów bez jakichkolwiek zmian w kodzie źródłowym projektu zintegrowano serwis Docker. Zostaną one dokładniej opisane w tej sekcji, a szczegóły dotyczące samej implementacji i jej ogólności poruszone w sekcji 3.7.8. Prezentację interfejsu wszystkich serwisów umieszczone w sekcjach 5.1.1 i 5.1.3.

3.5.1. Jupyter Notebook

Jupyter jest serwisem pozwalającym na interaktywną edycję i uruchamianie tak zwanych Notebooków. Notebooki to materiały, w których można zamieścić między innymi instrukcje, obrazy, tabele, a także kod w jednym z obsługiwanych języków programowania (np. Python lub Julia). Jupyter jest popularnym rozwiązaniem wykorzystywany do prowadzenia zajęć

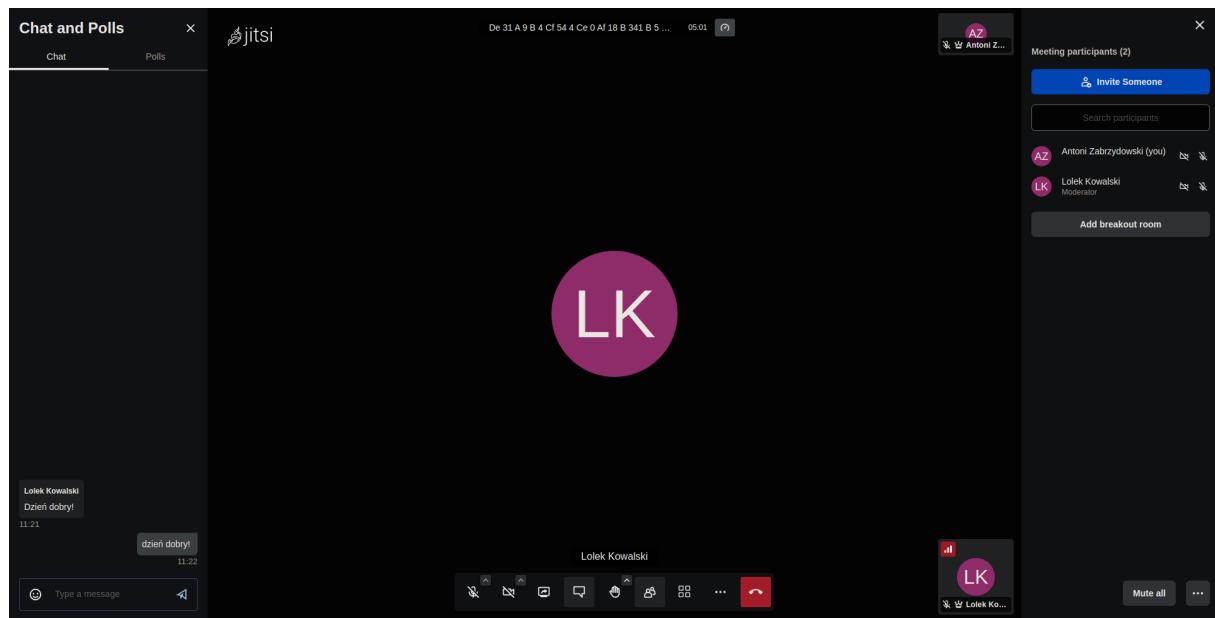
zarówno zdalnych, jak i stacjonarnych i sprawdza się dobrze, gdy osoba prowadząca ma przygotowany pewien skrypt, według którego chciałaby przeprowadzić zajęcia. Notebook może być reprezentacją takiego skryptu, która prowadzi studenta krok po kroku po przygotowanych przez prowadzącego przykładach. Zaletą tego rozwiązania jest możliwość wykonywania zadań w indywidualnym tempie przez każdego studenta, dzięki czemu mogą się oni wykazać i zdobyć praktyczne podczas wykonywania ćwiczenia.

Sposób w jaki Jupyter został zintegrowany w tym projekcie pozwala na dodanie przez nauczyciela własnego Notebooka, który zostanie następnie dostarczony do każdego środowiska, w którym będą pracować studenci. Po zakończeniu laboratoriów wyniki zostaną automatycznie zebrane z maszyn wirtualnych i udostępnione uczestnikom przez *Aplikację Webową*.

3.5.2. Jitsi Meet

Jitsi Meet to oprogramowanie służące do obsługi wideokonferencji oraz pełniące rolę komunikatora internetowego. Nauczyciel może wybrać dołączenie modułu z telekonferencją do dowolnych zajęć. Następnie w odpowiednim czasie przed samym rozpoczęciem spotkania zarówno on, jak i uczniowie otrzymują odpowiednio spreparowane linki. W celu dołączenia do sesji użytkownik musi jedynie kliknąć w dostarczony link, nie musi się logować ani wprowadzać żadnych danych. Rysunek 20 przedstawia interfejs serwisu Jitsi Meet, jako najważniejsze funkcjonalności systemu możemy wymienić:

- wideokonferencję z możliwością użycia kamery i mikrofonu,
- opcję udostępnienia ekranu,
- czat.



Rysunek 20: Interfejs serwisu Jitsi Meet z podłączonymi dwoma osobami

Do integracji serwisu wykorzystano kod z projektu Sozisel [13], w którym to cała aplikacja Jitsi Meet jest już odpowiednio skonfigurowana i gotowa do zaplanowania i uruchomienia wideokonferencji. Po uruchomieniu instancji maszyny wirtualnej z przygotowanym Soziselem, *Orkiestrator* wysyła do niej zapytania GraphQL mające na celu zarejestrować nauczyciela jako

nowego użytkownika, zaplanować oraz uruchomić sesję Jitsi, a na koniec wygenerować token spotkania, na podstawie którego generowane są linki umożliwiające dołączenie do konferencji.

3.5.3. QuizApp

QuizApp jest systemem pozwalającym na przeprowadzanie quizów. W celu utworzenia na jego podstawie modułu lekcyjnego poza typową konfiguracją wymagań sprzętowych nauczyciel techniczny musi podać długość quizu oraz listę pytań i odpowiedzi. Moduł jest uruchamiany na maszynie wirtualnej przypisanej do nauczyciela, wszyscy uczniowie dostają dedykowane linki (przypisane do ich adresu email) umożliwiające udzielenie odpowiedzi. Gdy uczeń odpowie na wszystkie pytania lub gdy upłynie czas quizu wyniki są zapisywane, a następnie udostępniane nauczycielowi w formie pliku YAML. Plik z wynikami zawiera przedstawione w czytelny sposób adresy email uczniów, uzyskane wyniki oraz odpowiedzi jakich udzielili na zadane pytania. Serwis przesyła wyniki jedynie do nauczyciela, jeśli będzie on chciał powiadomić o nich uczniów będzie musiał zrobić to ręcznie.

Jest to bardzo prymitywny serwis, pozwalający jedynie na przeprowadzenie quizu z pytaniami zamkniętymi, jednak dzięki temu jest też bardzo prosty w obsłudze dla użytkowników. Serwis został znaleziony na platformie GitHub, jest utrzymywany przez jednego programistę, a jego integracja miała na celu pokazanie, że nasz system jest rozwijalny również o takie projekty. Trzeba zaznaczyć, że nic nie stoi na przeszkodzie integracji kolejnego, zupełnie innego serwisu pozwalającego na przeprowadzenie quizów, nauczyciel techniczny miałby wówczas wybór i mógłby lepiej dobrać narzędzie do zastosowania (QuizApp wciąż może okazać się lepszym wyborem gdy jedyne czego potrzeba to prostego quizu z pytaniami zamkniętymi).

Rysunek 21 przedstawia przykładowy quiz, jak widać nie ma tu żadnych niepotrzebnych elementów (typu menu w górnym pasku), jedynie pytania, licznik czasu oraz opcja przejścia dalej. Przypisanie quizu do konta użytkownika odbywa się na podstawie losowego klucza w linku, bez linku nie ma możliwości wzięcia udziału w quizie, dzięki czemu mamy pewność że jeden uczeń w sposób niepowołany nie odpowie na pytania za innego ucznia, bądź uzyska dostęp do quizu zanim nauczyciel na to zezwoli. Rysunek 22 przedstawia fragment pliku z wynikami quizu, do którego dostęp ma nauczyciel po jego zakończeniu.

Rysunek 21: Jedno z pytań w przykładowym quizie w serwisie QuizApp

```

Włodzimierz Biały (wbialy@gmail.com):
  score: 2
  results:
    - id: 0
      question: Jaki jest wynik obliczenia 9 + 10?
      user_answer: 21
      # kolejne pytania i odpowiedzi...
Marek Naborzny (mnaborzny@gmail.com):
  score: 1
  # pytania i odpowiedzi kolejnego ucznia...

```

Rysunek 22: Przykładowy plik z wynikami quizu z serwisu QuizApp

3.5.4. Docker

Docker jest oprogramowaniem dostarczającym wirtualizacji na poziomie systemu operacyjnego w postaci tzw. kontenerów. Obraz Dockera, zawierający kod źródłowy utworzonej aplikacji wraz z jej zależnościami, można uruchomić na dowolnym systemie, na którym działa Docker. Platforma Docker Hub¹⁴ jest publicznym repozytorium obrazów Dockera z różnym oprogramowaniem takim jak bazy danych, czy nawet nasz projekt. Każdy taki obraz można pobrać, a następnie uruchomić w celu wykorzystania dostarczanych przez niego funkcjonalności.

Zarówno konfiguracja, jak i tworzenie nowych obrazów Dockera może nie być prostym zadaniem i prawdopodobnie serwis mógłby znaleźć zastosowanie jedynie w placówkach, których pracownicy mają umiejętności z zakresu informatyki. Serwis dostarcza jednak bardzo szerokiego zakresu funkcjonalności dla takich placówek. Bez jakichkolwiek zmian w kodzie źródłowym projektu można uruchomić moduł oparty na serwisie Docker i następnie wykorzystać jego funkcjonalności po wejściu w dostarczony link. Przykładowe zastosowanie, wraz ze wskazaniem sposobu konfiguracji opisano w sekcji 5.1.3. Serwis pozwala na dołączenie dowolnych plików z poziomu interfejsu *Aplikacji Webowej*, a także na zapisanie wyników po zakończeniu zajęć, może zajść jednak potrzeba modyfikacji wykorzystanego obrazu, ponieważ zakładamy, że wszystkie wyniki będą dostępne w jednym folderze. Nowe obrazy utworzone przez nauczycieli technicznych muszą być publicznie dostępne w repozytorium Docker Hub. Serwis obsługuje zarówno tryb izolowany jak i współdzielony, jednak w zależności od obrazu użycie jednego z nich może nie mieć większego sensu.

Możliwość uruchomienia dowolnego oprogramowania może nieść za sobą pewne zagrożenia. Przykładowo w momencie kradzieży konta użytkownika z uprawnieniami nauczyciela technicznego, osoba atakująca mógłby utworzyć moduł będący w stanie „kopać” kryptowaluty, a następnie uruchomić go na wielu instancjach maszyn wirtualnych, przez co placówka mógłby ponieść zauważalne koszty. Z tego powodu konieczne jest odpowiednie monitorowanie działania systemu przez administratora, a w razie większych obaw możliwe jest wyłączenie opcji tworzenia modułów bazujących na tym serwisie.

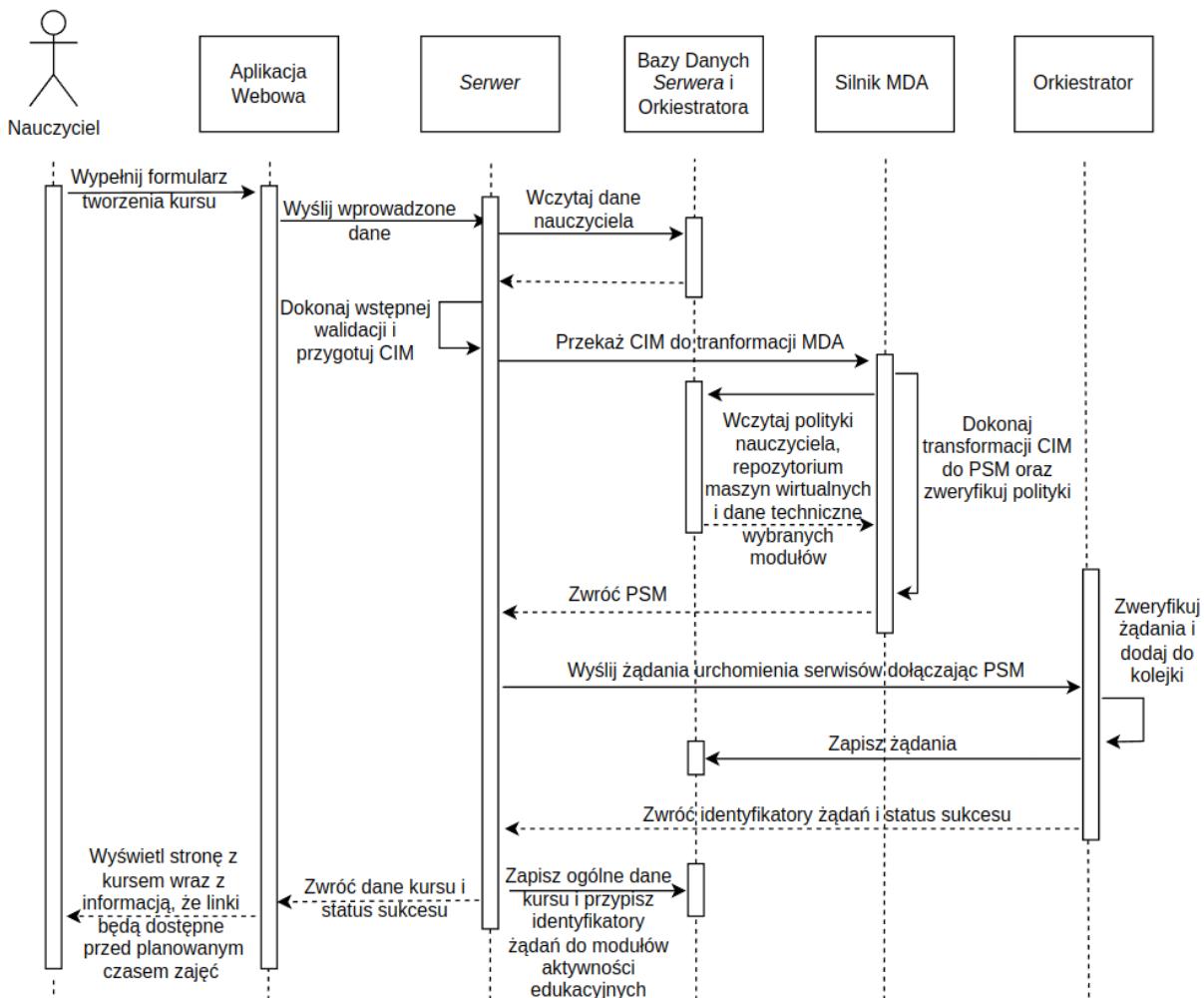
¹⁴Lista obrazów w repozytorium Docker Hub: <https://hub.docker.com/search?q=&type=image>

3.6. Zasady działania ważniejszych elementów logiki biznesowej

W sekcji omówione zostaną aspekty realizacji kluczowych z punktu widzenia użytkownika funkcjonalności. Zostaną zaprezentowane i opisane diagramy sekwencji (ang. sequence diagrams) tłumaczące przepływ logiki i danych w systemie. Przedstawione opisy będą opierać na dosyć wysokim poziomie abstrakcji, szczegóły techniczne ważniejszych elementów wyjaśniono w sekcji 3.7. Dla zwiększenia czytelności na diagramach bazy danych *Serwera* i *Orkiestratora* traktowane są jako jeden obiekt, jeśli *Orkiestrator* dokonuje na nim operacji to w rzeczywistości jest to baza danych *Orkiestratora*, w przeciwnym przypadku, baza danych *Serwera*.

3.6.1. Tworzenie kursu

Moduły lekcyjne, będące skonfigurowanymi przez nauczyciela technicznego serwisami, mogą zostać uruchomione w ramach aktywności edukacyjnych wchodzących w skład kursu. Pierwszym etapem związanym z przeprowadzaniem zajęć jest utworzenie przez nauczyciela kursu. Diagram 23 przedstawia poszczególne etapy tego procesu.



Rysunek 23: Diagram sekwencji tworzenia kursu

Proces zaczyna się od *Aplikacji Webowej*, w której nauczyciel musi wybrać opcję tworzenia kursu. Wyświetlony zostaje formularz, który zawiera pola takie jak nazwa kursu, jego opis oraz

pole potrzebne *Silnikowi MDA* - planowaną liczbę uczniów. Pole to jest kluczowe, ponieważ w momencie tworzenia kursu nauczyciel nie dodaje do niego studentów, może to zrobić później ręcznie lub wysłać im link do dołączenia aby zrobili to sami. Planowana liczba studentów jest jednak niezbędna, żeby odpowiednio zaplanować wykorzystanie maszyn wirtualnych. Skutki wprowadzenia zbyt dużej bądź zbyt małej liczby zostaną opisane w dalszej części.

W kolejnym kroku *Aplikacja Webowa* generuje podrzędny formularz umożliwiający skonfigurowanie aktywności edukacyjnych wchodzących w skład kursu. Dla każdej aktywności nauczyciel musi wprowadzić czas jej odbycia oraz moduły lekcyjne, które mają wchodzić w jej skład. Nauczyciel ma możliwość wybrania ich z listy dostępnych publicznych modułów utworzonych przez nauczycieli technicznych lub z własnych modułów, jeśli sam pełni tę rolę.

Wprowadzone dane są wysyłane do *Serwera*, który w pierwszej kolejności sprawdza czy są one poprawne (przykładowo czy aktywność nie trwa za długo lub czy nie jest zaplanowana na czas przeszły). Następnie z danych tworzony jest CIM, który zgodnie z opisem w sekcji 3.4 zostaje przetłumaczony na PSM w *Silniku MDA*.

Jeśli polityki nie zostały spełnione bądź wykryto błędy we wprowadzonych danych tworzenie kursu zostanie przerwane, a błędy zostaną zwrócone i wyświetlane użytkownikowi w celu umożliwienia poprawy. Jeśli wszystko poszło pomyślnie *Serwer* tworzy żądania uruchomienia serwisów o określonych przez nauczyciela porach i wysyła je do *Orkiestratora*, który dokonuje ostatecznej walidacji czy będzie możliwe ich uruchomienie. *Orkiestrator* zapisuje żądania w bazie danych i dodaje je do kolejki umożliwiającej ich obsługę we właściwym czasie. Na podstawie zwróconych identyfikatorów *Serwer* tworzy w bazie danych powiązania między modułami wchodzącymi w skład poszczególnych aktywności, uczestnikami kursu oraz identyfikatorami żądań. Ma to na celu umożliwienie późniejszego przypisania linków, instrukcji podłączenia oraz wyników prac do właściwych kont uczestników kursu.

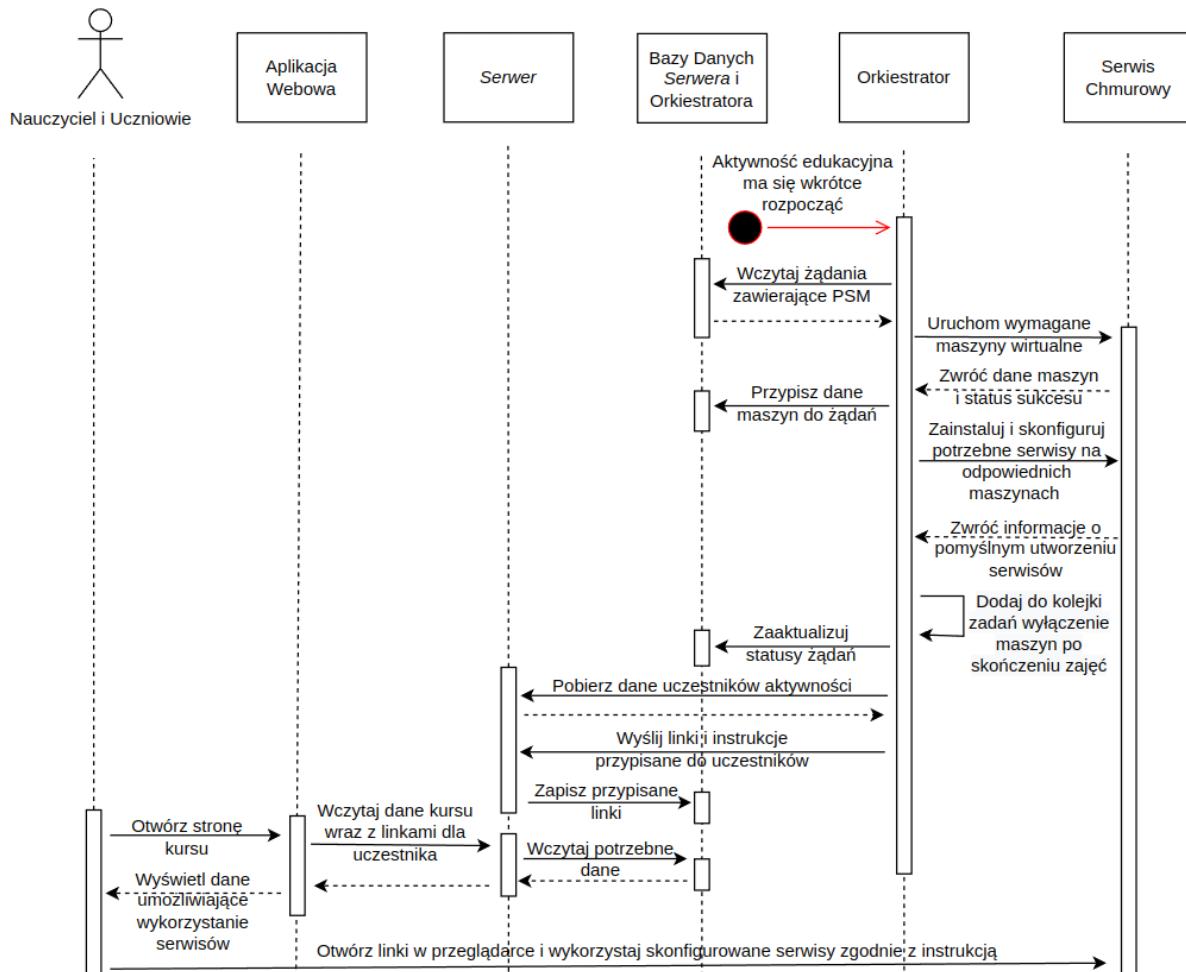
Do *Aplikacji Webowej* zwrócone zostają informacje, na podstawie których wyświetlana jest nauczycielowi strona utworzonego kursu, przy widokach aktywności edukacyjnych znajdują się informacje, że linki do serwisów będą dostępne w zaplanowanym czasie. Nauczyciel może dodawać dodatkowe aktywności edukacyjne po utworzeniu kursu, ich obsługa jest bardzo podobna. Możliwa jest również edycja ogólnych danych kursu i usuwanie aktywności edukacyjnych, które polega na anulowaniu wysłanych do *Orkiestratora* żądań.

3.6.2. Przebieg aktywności edukacyjnej

Wszystkie potrzebne serwisy muszą być dostępne w zaplanowanym przez nauczyciela momencie rozpoczęcia aktywności edukacyjnej. Diagram 24 w znacznym uproszczeniu przedstawia ten proces.

Proces zaczyna się chwilę przed rozpoczęciem aktywności edukacyjnej, dokładny czas jest zależny od wykorzystanych serwisów, im dłużej trwa uruchomienie serwisu tym wcześniej zostanie podjęta akcja, tak by wszystko było gotowe o zaplanowanej godzinie. *Orkiestrator* na podstawie wysłanych przy tworzeniu kursu żądań uruchamia równolegle wszystkie potrzebne maszyny wirtualne dostępne w infrastrukturze dostawcy usług chmurowych. Gdy maszyny zostaną utworzone *Orkiestrator* przystępuje do instalowania i konfiguracji potrzebnych serwisów, ta akcja również dzieje się równolegle dla wszystkich maszyn, dokładne działanie opisano w sekcjach 3.7.6 i 3.7.7.

Po udanym utworzeniu serwisów *Orkiestrator* wysyła do *Serwera* zapytanie o dane uczestników kursu, jest to konieczne, ponieważ niektóre serwisy używają linków do konfiguracji dostępu do serwisu, przykładowo Jitsi Meet ustawia w linku imię i nazwisko uczestnika wideokonferencji, dodatkowo potrzebna jest jego rola, żeby tylko nauczyciel miał większe uprawnienia



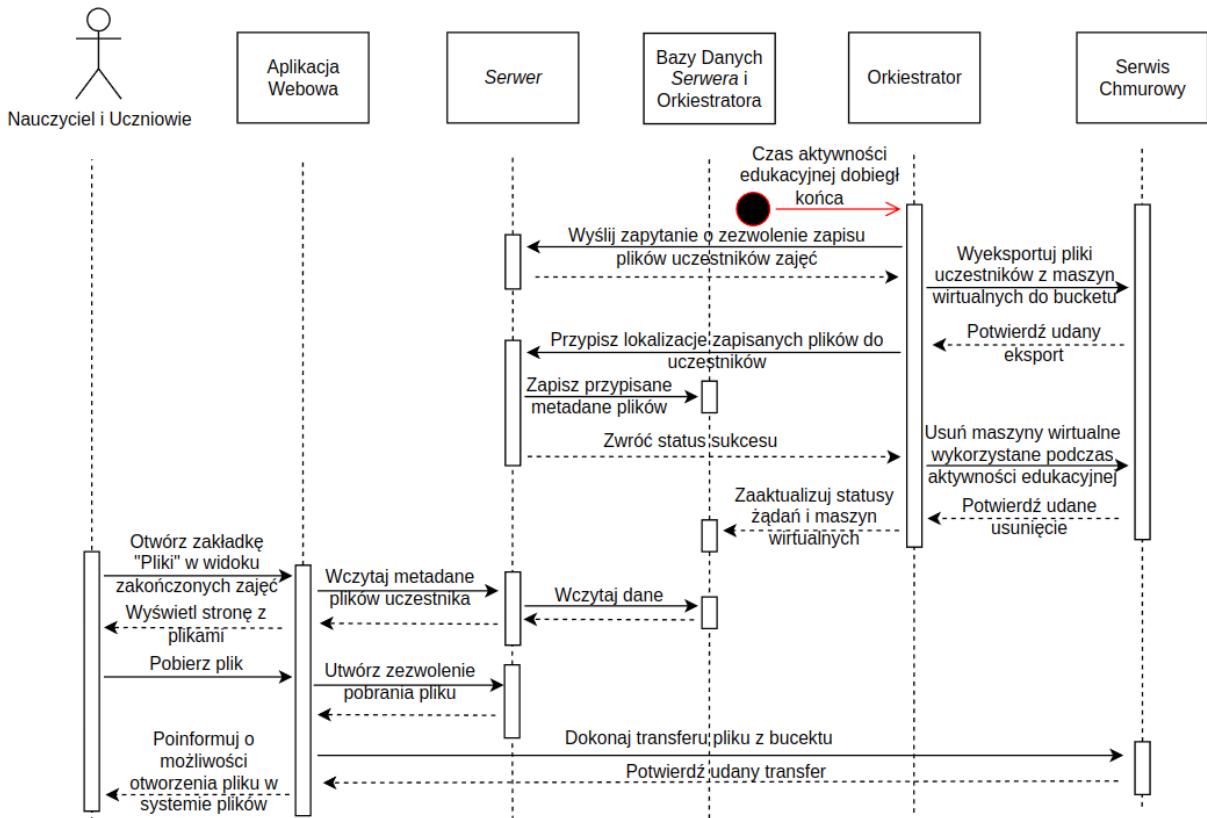
Rysunek 24: Diagram sekwencji uruchamiania serwisów

(typu wyciszenie wszystkich uczestników). Na podstawie tych danych *Orkiestrator* tworzy linki i wysyła je do *Serwera* wraz z informacją, który link powinien zostać przypisany do którego uczestnika, *Serwer* zapisuje je w bazie danych.

Po zapisaniu linków uczestnicy będą mogli uzyskać do nich dostęp po odświeżeniu strony kursu w *Aplikacji Webowej*. Nauczyciel ma opcję wymagania potwierdzenia przed wysłaniem linków do danego serwisu do uczniów, w przypadku wybrania tej opcji linki zostaną wysłane dopiero po ręcznym potwierdzeniu. Ma to zastosowanie przy serwisach typu quiz, nauczyciel może chcieć uruchomić quiz dopiero blisko końca zajęć a nie od samego początku, żeby uczniowie nie mieli wcześniej dostępu do pytań.

Wraz z linkami wyświetlane są instrukcje umożliwiające skorzystanie z serwisu, przykładowo dla Jupyter w instrukcji znajdzie się hasło oraz informacja gdzie je wpisać. Uczestnicy korzystając z tych danych mogą uzyskać dostęp do serwisów za pośrednictwem przeglądarki internetowej. Aktywność edukacyjna polega na wykorzystaniu serwisów zgodnie z ich przeznaczeniem, z reguły nadanym przez nauczycieli technicznych, którzy je skonfigurowali.

Po zakończeniu planowanego czasu zajęć konieczne jest usunięcie maszyn wirtualnych, żeby nie ponosić dodatkowych kosztów. Niektóre serwisy umożliwiają zapis wyników prac uczestników więc należy to również uwzględnić, diagram 25 przedstawia uproszczony schemat kończenia aktywności edukacyjnej.



Rysunek 25: Diagram sekwencji kończenia aktywności edukacyjnej

Orkiestrator posługując się kolejką zadań wykrywa, że nadszedł czas zakończenia aktywności edukacyjnej. Jeśli wykorzystane zostały serwisy, które obsługują opcję zapisu prac uczniów (przykładowo Jupyter obsługuje, a Jitsi Meet nie) to *Orkiestrator* rozpoczyna proces eksportu plików z maszyn wirtualnych, żeby nie zostały one utracone, dokładny proces transferu opisano w sekcji 3.7.2. W przypadku nieudanego eksportu *Orkiestrator* dostarcza logi z informacją o porażce, administrator ma 2 godziny na zrobienie tego ręcznie, w przeciwnym przypadku pliki zostaną utracone. Z uwagi na fizyczną lokalizację maszyn wirtualnych i systemu odpowiedzialnego za przechowywanie plików (kubelka) w jednym regionie prawdopodobieństwo takiego zdarzenia jest bardzo małe i jest dodatkowo redukowane dzięki powtarzaniu prób eksportu plików.

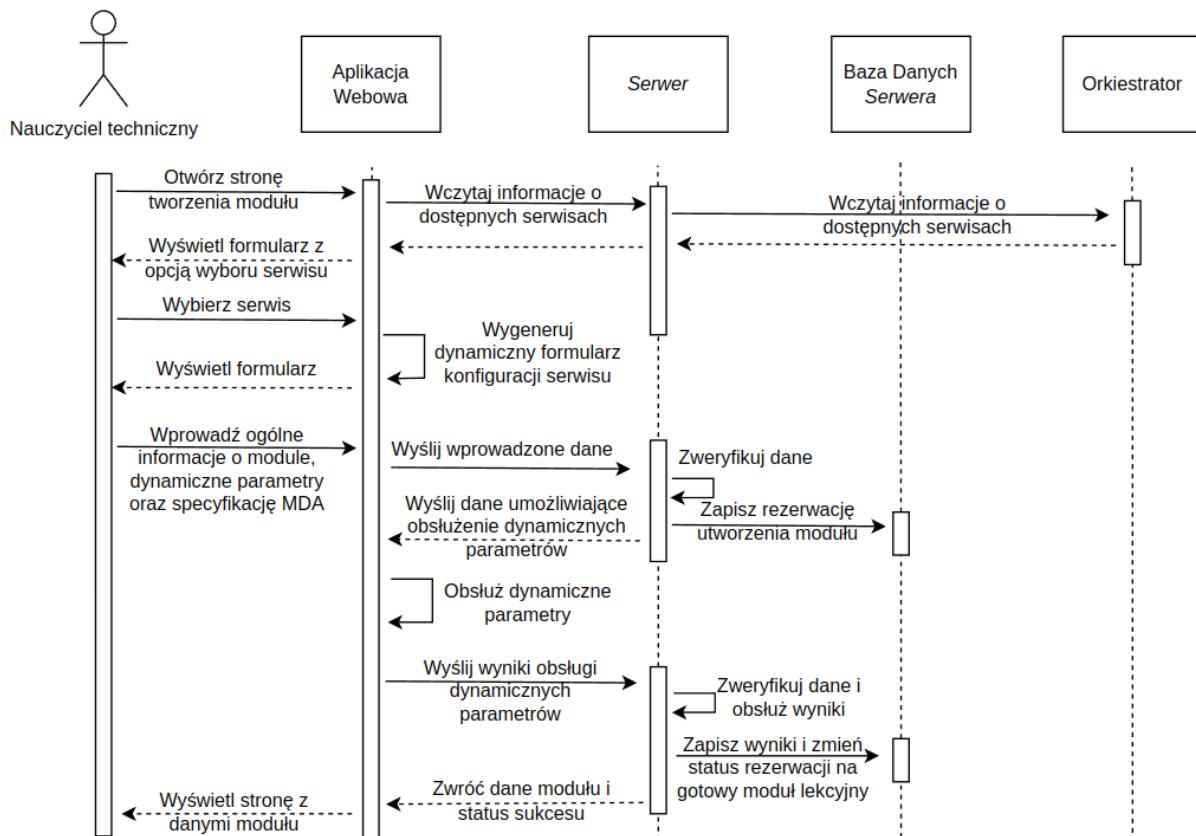
Po udanym eksportie *Orkiestrator* usuwa wszystkie maszyny wirtualne wykorzystane do przeprowadzenia danej aktywności edukacyjnej. Uczestnicy kursu tracą możliwość dostępu do nich, nie zaimplementowano opcji przedłużania czasu trwania zajęć, gdyż nie było to priorytetem. Uczniowie mają dostęp do swoich plików z zajęć poprzez *Aplikację Webową*, która umożliwia ich pobranie, nauczyciel ma dostęp zarówno do własnych plików jak i do plików wszystkich uczniów.

We wcześniejszej sekcji zwrócono uwagę na parametr podawany przez nauczyciela podczas tworzenia kursu - planowaną liczbę uczestników. Jeśli liczba ta będzie zbyt duża (przykładowo pewien student nie zdąży dołączyć do aktywności przed jej rozpoczęciem) to maszyna wirtualna dla niego nie zostanie uruchomiona, jeśli dołączy chwilę po rozpoczęciu zajęć to nie dostanie linków, oczywiście również wyniki prac nie zostaną zapisane. Taki uczeń będzie mógł wziąć jednak udział we wszystkich kolejnych aktywnościach edukacyjnych. W przypadku podania zbyt małej liczby (przykładowo uczeń przepisał się z jednej grupy bądź klasy do innej) ko-

nieczne byłoby odwołanie aktualnie zaplanowanych zajęć, ponowne obliczenie transformacji MDA i wysłanie poprawionych żądań do *Orkiestratora*. Z uwagi na brak czasu i niski priorytet tej funkcjonalności nie zostało to ostatecznie zaimplementowane, aczkolwiek wszystkie potrzebne mechanizmy typu odwoływanie żądań są gotowe, więc byłoby to możliwe. Na ten moment maszyny wirtualne dla nadmiarowych studentów nie zostaną utworzone i nie dostaną oni linków.

3.6.3. Tworzenie modułów lekcyjnych

Praktyczność systemu jest zależna od dostępności modułów lekcyjnych konfigurowanych przez nauczycieli technicznych, jak już wcześniej zaznaczono zwykły nauczyciel też może pełnić tę rolę. Nadrzędnym celem było umożliwienie integracji nowych, początkowo nieprzewidzianych serwisów, bez ingerencji w kod źródłowy całego projektu i w model danych. Diagram 26 obrazuje przebieg procesu tworzenia modułu lekcyjnego.



Rysunek 26: Diagram sekwencji tworzenia modułu lekcyjnego

Orkiestrator jest podsystemem odpowiedzialnym za obsługę serwisów, tylko on wie jakie serwisy są dostępne oraz jakich parametrów wymagają do konfiguracji, z tego powodu wyświetlenie formularza z tworzeniem modułu wymaga komunikacji z *Orkiestratorem*, *Serwer* w tym przypadku pełni jedynie rolę pośrednika.

Na podstawie zwróconych danych *Aplikacja Webowa* może wygenerować dynamiczny formularz, zawierający różne pola w zależności od wybranego serwisu. Przykładowo dla Jupytera należy wprowadzić plik z początkowym notebookiem, a dla quizu jego czas trwania oraz plik z pytaniami. Dane są wysyłane do *Serwera* w celu weryfikacji, proces tworzenia może wymagać dodatkowych akcji po stronie *Aplikacji Webowej* dlatego nie jest możliwa jego atomowa

realizacja, przykładową akcją jest przesłanie pliku do serwisu chmurowego. Z tego powodu w pierwszym kroku tworzona jest rezerwacja, żeby inny użytkownik nie mógł jednocześnie stworzyć modułu o tej samej nazwie powodując niespójność w bazie danych, moduł będący rezerwacją nie jest nigdzie wyświetlany ani możliwy do użycia.

Serwer jako wynik rezerwacji zwraca dane umożliwiające dokończenie obsługi dynamicznych parametrów po stronie klienta, przykładowo dla pliku wysyłane jest zezwolenie przesłania go do serwisu chmurowego. *Aplikacja Webowa* po skończeniu obsługi wysyła wyniki do *Serwera*, który je weryfikuje na podstawie informacji o serwisie, wcześniej zwróconych od *Orkiestratora*. Jeśli wszystkie dane są poprawne moduł jest zapisywany jako gotowy do wykorzystania. Rezerwacje, których nie udało się potwierdzić w przeciągu jednego dnia są automatycznie usuwane. Dokładna obsługa dynamicznych parametrów została opisana w sekcji 3.7.8.

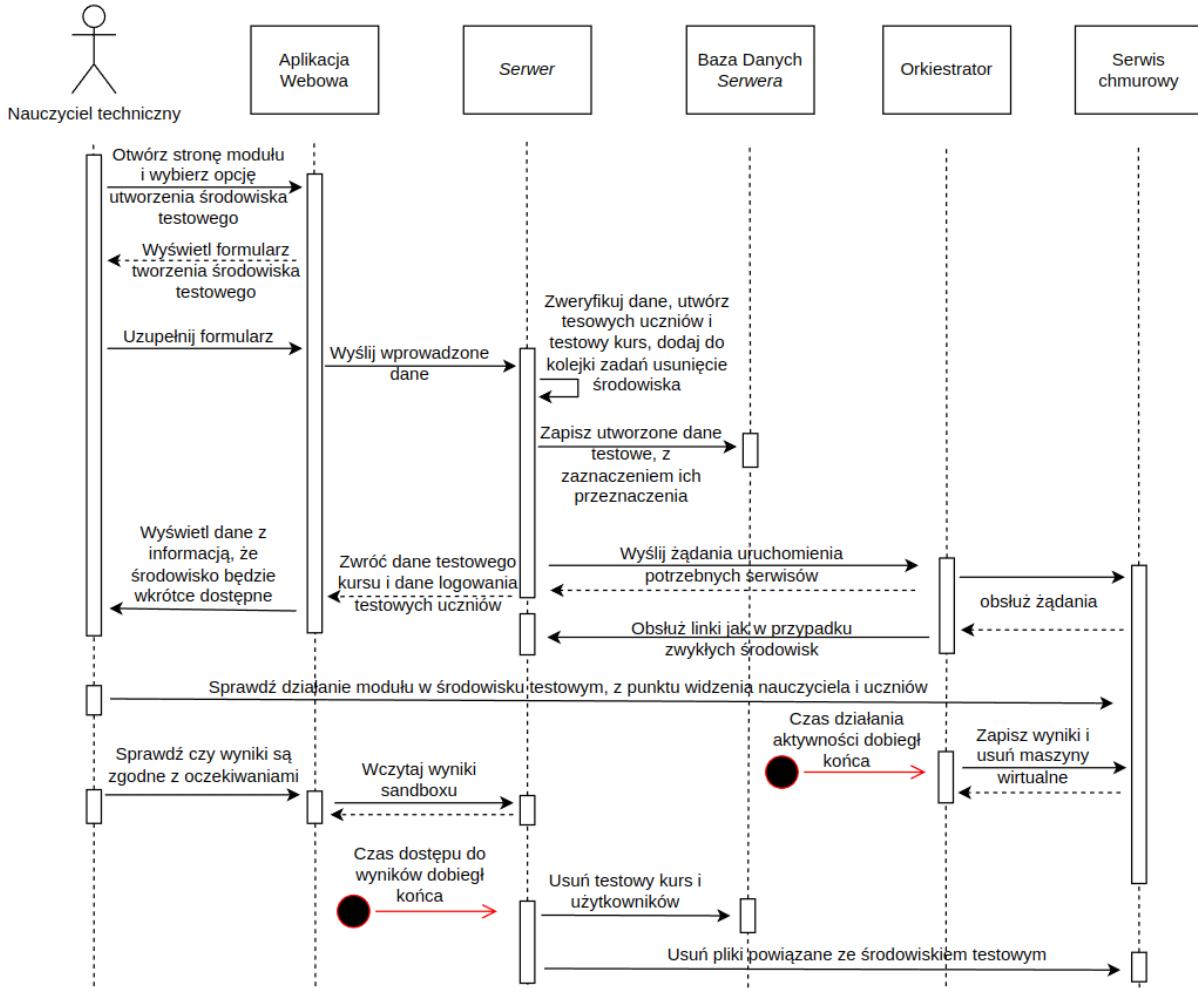
Jednym z parametrów wprowadzanych przez nauczyciela technicznego jest ustalenie czy moduł jest prywatny czy publiczny. Prywatny moduł jest możliwy do wykorzystania tylko przez jego twórcę, publiczny mogą wykorzystać wszyscy nauczyciele. Nauczyciel techniczny może edytować w późniejszym czasie wszystkie dane modułu, jednak przy zmianie modułu publicznego na prywatny nauczyciele, którzy już go wybrali do aktywności edukacyjnych wciąż będą mogli go wykorzystać. Wyjątkiem są typy serwisu oraz jego izolacji, z przyczyn technicznych nie można ich zmienić.

3.6.4. Obsługa środowisk testowych

Weryfikacja wprowadzonych parametrów modułu polega na sprawdzeniu jedynie czy są one poprawne pod kątem semantycznym (na przykład czy nie wpisano ujemnej liczby wirtualnych procesorów maszyny). Nigdzie nie ma gwarancji, że utworzony moduł będzie działał poprawnie, przykładowo maszyna wirtualna o zbyt niskich parametrach może nie być w stanie obsłużyć notebooka Jupytera wymagającego wykonywania kosztowych obliczeń z dziedziny uczenia maszynowego. Żeby umożliwić nauczycielowi technicznemu sprawdzenie poprawności działania modułu zaimplementowano mechanizm obsługi środowisk testowych (tzw. sandboxów), diagram 27 w znacznym uproszczeniu przedstawia zasadę ich funkcjonowania.

Środowisko testowe jest automatycznie utworzonym kursem z jedną aktywnością edukacyjną. Aktywność edukacyjna wykorzystuje pojedynczy moduł, który nauczyciel techniczny chce przetestować, czas rozpoczęcia aktywności jest zależny od serwisu, w ogólności rozpoczęcie się ona najszybciej jak to możliwe. Czas dostępu do maszyn wirtualnych oraz do wyników modułu jest konfigurowalny przez twórcę środowiska testowego w formularzu w *Aplikacji Webowej*. Dodatkowo tworzone są testowe konta uczniów, a ich dane logowania udostępniane są nauczycielowi technicznemu, pozwala to na przetestowanie modułów współdzielonych.

Samo tworzenie kursu i przebieg aktywności edukacyjnej jest identyczny jak opisano w sekcjach 3.6.1 i 3.6.2. Po zakończeniu aktywności nauczyciel ma wgląd do plików przypisanych do utworzonych kont przez określony czas, po którym wszystkie testowe dane zostają automatycznie usunięte, ich ręczna edycja nie jest możliwa w żadnym etapie działania środowiska.



Rysunek 27: Diagram sekwencji działania środowisk testowych

Funkcjonalność środowisk testowych ma zastosowanie nie tylko do testowania modułów przez nauczycieli technicznych, ale także przy wyborze publicznych modułów do zajęć przez nauczycieli. Mogą oni dokonać wyboru nie tylko na podstawie opisu i instrukcji wykorzystania modułu ale również mogą wstępnie sprawdzić w środowisku testowym czy spełnia on oczekiwania. Ponadto daje to im opcję lepszego przygotowania się do zajęć i napisania instrukcji dla uczniów. Liczba środowisk testowych możliwych do jednoczesnego uruchomienia przez danego użytkownika może zostać ograniczona przez administratora, domyślnie wynosi ona 1, ma to na celu redukcję kosztów związanych z działaniem maszyn wirtualnych.

3.7. Szczegóły implementacyjne warstwy technicznej

W sekcji zostaną doprecyzowane implementacyjne aspekty realizacji ważniejszych funkcjonalności systemu, do których do tej pory odnoszono się w sposób abstrakcyjny. Opisy obejmą wszystkie podsystemy i interakcje między nimi, mając na celu umożliwienie osobom niezaznajomionym z kodem źródłowym zrozumienie jego działania i założeń jakie ma spełniać. Część udokumentowanych tutaj funkcji będzie stanowić podstawę wytycznych dotyczących administracji systemem, przedstawionych w sekcji 3.8.

3.7.1. Bezpieczeństwo systemu

W sekcji zostanie dokładnie opisane w jaki sposób zagwarantowano bezpieczeństwo systemu. Omówione zostaną wszystkie kluczowe mechanizmy, od momentu tworzenia konta użytkownika, przez zmianę hasła, logowanie i zarządzanie sesją, aż po zabezpieczenie samej komunikacji HTTP i przyznawanie dostępu do konkretnych zasobów. Bezpieczeństwo przechowywania plików oraz komunikacji między *Serwerem* a *Orkiestratorem* opisano odpowiednio w sekcjach 3.7.2 i 3.7.3.

a) Tworzenie konta użytkownika

Funkcjonalność tworzenia konta jest dostępna jedynie dla administratora, ponieważ zgodnie z wymaganiami system miał być zamknięty dla ograniczonego i sprecyzowanego grona osób - pracujących bądź uczących się w danej placówce. Administrator posługując się interfejsem webowym konfiguruje role oraz dane każdego użytkownika - w szczególności ich adresy e-mail, w zamyśle powinny one znajdować się w domenie placówki, jednak nie jest to konieczne. Administrator nie podaje i nie może odczytać haseł użytkowników, zostają one wygenerowane automatycznie przez system, jako silne pseudolosowe ciągi znaków, a następnie poddane hashowaniu (wyliczeniu kryptograficznej funkcji skrótu). Jako funkcję skrótu wybrano algorytm scrypt, ponieważ jest on bardzo wymagający obliczeniowo. Dzięki temu ewentualne ataki siłowe (ang. bruteforce) są dla atakującego bardzo kosztowe czasowo, z drugiej strony użytkownik znający hasło będzie zwykle musiał je wpisać tylko jeden raz, więc zwiększyony czas obliczeń nie jest problemem.

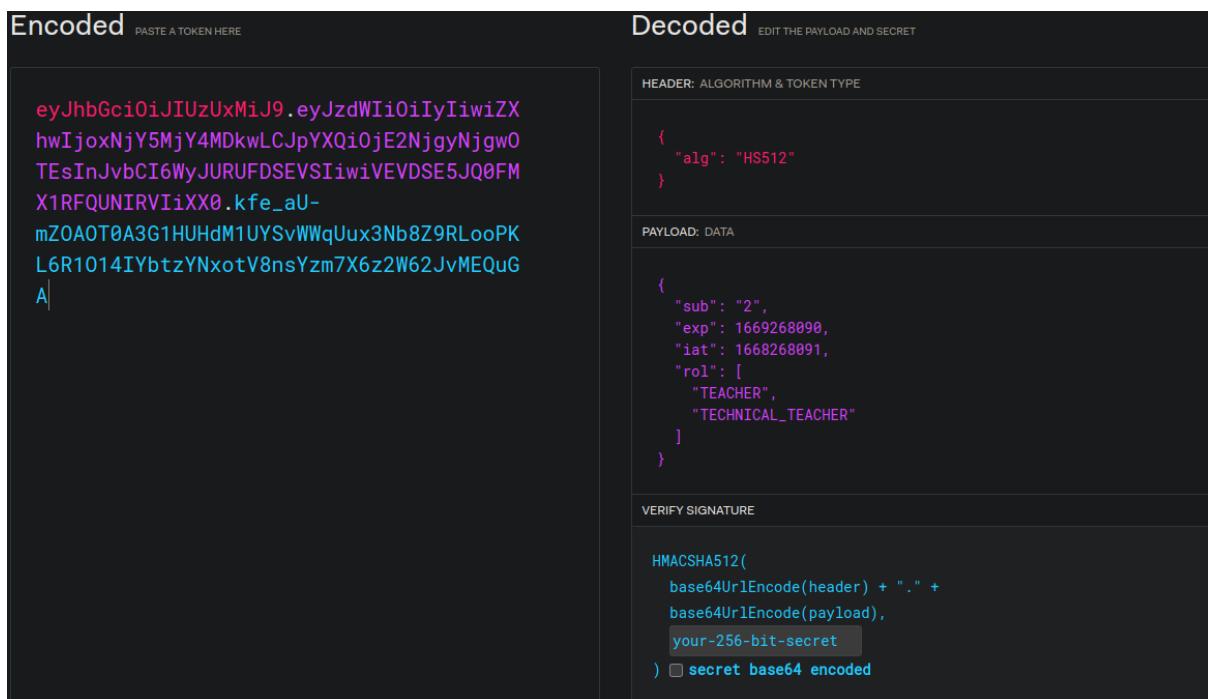
Aby uzyskać dostęp do konta użytkownik musi jedynie mieć dostęp do swojego e-maila. Z poziomu interfejsu webowego użytkownik musi wybrać opcję przypominania hasła, po czym na wskazany adres e-mail zostanie wysłany token zmiany hasła (również losowy ciąg znaków, zmieniany po każdej zmianie hasła). Użytkownik musi przekopiować token do formularza na stronie i wpisać nowe hasło, następnie może przejść do logowania. W przypadku podania nieprawidłowego tokenu bądź za słabego hasła system wyświetla błąd. Hasło wymaga minimum 8 znaków, w tym minimum 2 specjalnych i 2 cyfr, ma to na celu zwiększenie odporności na ataki słownikowe.

b) Logowanie do systemu i zarządzanie sesją

Logowanie przez interfejs webowy polega na podaniu adresu e-mail oraz hasła. Dane są wysyłane do *Serwera*, który znajduje konto użytkownika posługując się mailem, oblicza funkcję skrótu podanego hasła i sprawdza czy jest ono identyczne z przechowywanym w bazie danych.

Po pomyślnym uwierzytelnieniu *Serwer* generuje token JWT, który zawiera dane użytkownika umożliwiające jego przyszłe uwierzytelnienie bez podania hasła. Rysunek 28 przedstawia przykładowy token, po lewej stronie jest on zakodowany metodą base64.

Token jest wysyłany w każdym żądaniu klienta, w nagłówku HTTP *Authorization*. Wiodoczne w sekcji *payload* dane umożliwiają sprawdzenie komu został nadany token oraz weryfikację ról w systemie. Żeby sprawdzić czy token został rzeczywiście wygenerowany po pomyślnym zalogowaniu użytkownika stosowany jest podpis cyfrowy. Serwer przy generowaniu tokenu podpisuje cyfrowo wszystkie dane w sekcji *payload* i dokleja je do tokenu (niebieski tekst na rysunku), następnie po otrzymaniu żądania od klienta weryfikuje podpis, jeśli jest zgodny to znaczy że został rzeczywiście wygenerowany przez Serwer, czyli właściciel tokenu musiał znać hasło (lub wykraść token, do czego przejdziemy za chwilę). Do realizacji podpisu cyfrowego wykorzystano algorytm HMAC z funkcją skrótu SHA512, wymaga to skonfigurowania na Serwerze tajnego klucza długości minimum 64 znaków.



Rysunek 28: JWT odpowiednio po zakodowaniu metodą base64, oraz przed zakodowaniem

Po stronie klienta token przechowywany jest w pamięci trwałe przeglądarki (Local Storage). Dostęp do tych danych jest możliwy z poziomu języka JavaScript bądź ręcznie za pomocą narzędzi developerskich. W związku z tym taki token może zostać wykradziony i używany do uwierzytelnienia przez niepowołaną osobę. Żeby ograniczyć skutki z tym związane token musi mieć ograniczony czas życia, w projekcie został skonfigurowany na 15 minut, po tym czasie, jeśli nie został odświeżony, wymagane będzie kolejne logowanie. Akcja ta może być jednak irytująca dla użytkownika, dlatego wprowadzono kolejny mechanizm - tokenów do odświeżania (ang. refresh tokens). Taki token jest podobny do wyżej opisanego JWT, jednak ma istotnie dłuższy czas życia, w systemie skonfigurowany na jeden dzień. Jeśli użytkownik będzie aktywny na stronie internetowej i token do odświeżania jeszcze nie wygasł to zwykły token dostępu zostanie automatycznie odświeżony przez wygaśnięciem, dzięki temu kolejne logowanie nie będzie konieczne.

Taki token jest również przechowywany w pamięci przeglądarki więc można by było sądzić, że tak samo może zostać wykradziony, w tym przypadku stosowane są jednak dodatkowe metody zabezpieczające. W przeciwieństwie do zwykłych tokenów, tokeny od-

świeżania są stanowe. Każdy wydany token jest zapisywany w bazie danych w tabeli **Refresh Tokens**, po wygaśnięciu (bądź ręcznym wylogowaniu użytkownika) jest usuwany. W momencie podejrzenia kradzieży sesji logowania użytkownik może usunąć tokeny odświeżania poprzez zmianę hasła, dzięki czemu dostęp do konta dla atakującego będzie ograniczony do względnie krótkiego czasu życia tokenu dostępu. Zysk z tego podejścia w stosunku do uwierzytelniania za pomocą ciasteczek (ang. cookies) polega głównie na wydajności, w przypadku ciasteczek serwer musi przy każdym żądaniu wykonywać zapytanie do bazy danych, w przypadku JWT wystarczy to zrobić przy odświeżaniu tokenu. Dodatkowo tokeny w przeciwieństwie do ciasteczek są odporne na ataki CSRF.

c) Autoryzacja użytkowników

Do autoryzacji wykorzystano mechanizm ról. Po pierwszym uruchomieniu systemu twozone jest automatycznie jedno konto administratora ze skonfigurowanym adresem e-mail, w celu otrzymania hasła konieczne jest jego zresetowanie. Administrator może tworzyć nowe konta i przyznawać im role, w szczególności również rolę administratora. Dostęp do każdego zasobu za pomocą żądań HTTP został ograniczony dla odpowiednich ról. Przykładowo na rysunku 29 przedstawiono, że odczyt danych publicznego kursu jest możliwy dla użytkownika z rolą Nauczyciela bądź Studenta, jednak tworzenie kursu jest możliwe tylko dla Nauczyciela. Stosowana jest hierarchia ról, to znaczy, że administrator ma jednocześnie prawa nadane każdej innej roli.

```
@GetMapping("/{summary}/{uuid}")
@PreAuthorize("hasAnyRole('TEACHER', 'STUDENT')")
public CourseSummaryDto getPublicCourseData(AccessToken token, @PathVariable String uuid) {
    // ...
}

@PostMapping()
@PreAuthorize("hasRole('TEACHER')")
public CourseDetailsDto addCourse(AccessToken token, @RequestBody CreateCourseRequest createCourseRequest) {
    // ..
}
```

Rysunek 29: Ograniczenia dostępu do zasobów dla ról

d) Poufność komunikacji między klientem a Serwerem

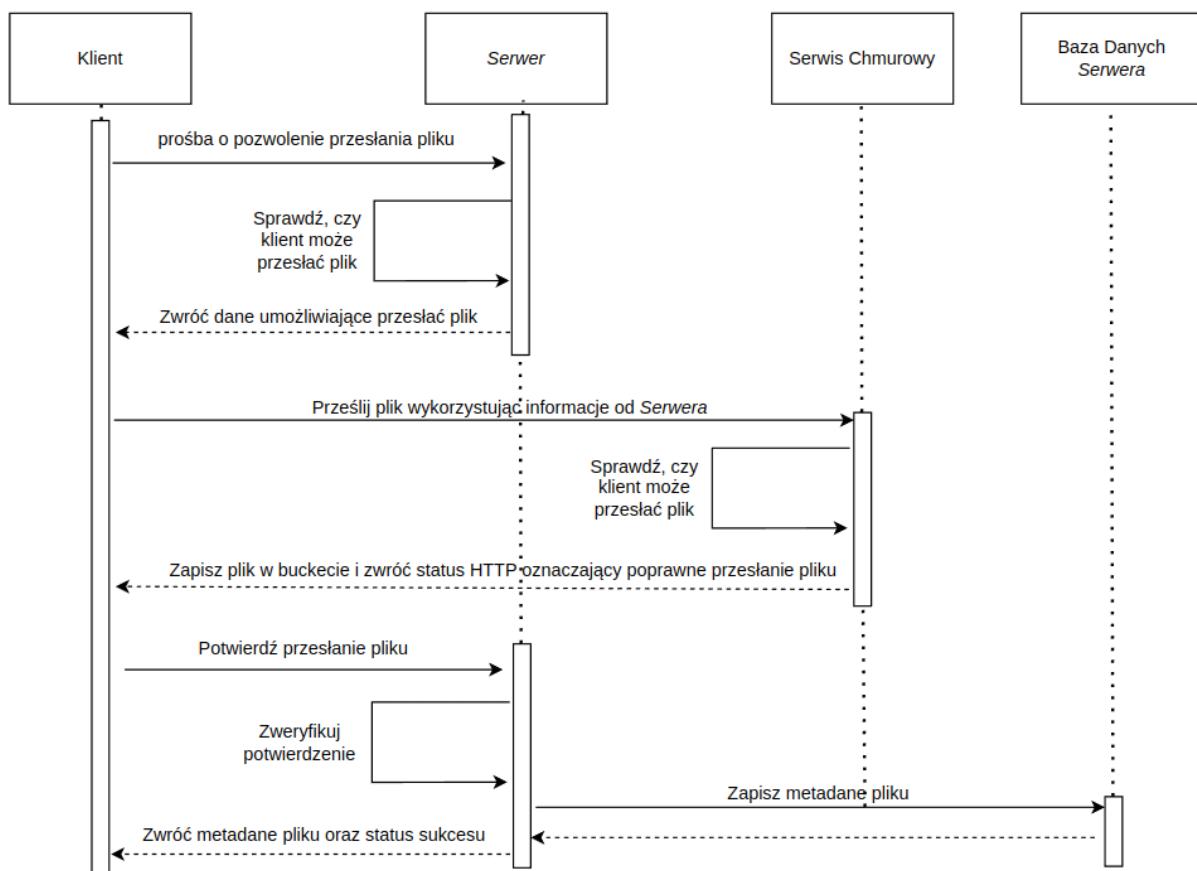
W celu uniemożliwienia stronom trzecim podsłuchiwanego komunikacji sieciowej, tym samym przykładowo kradzieży hasła logowania, należy stosować protokół HTTPS. W rzeczywistej placówce należałoby się ubiegać o nazwę domeny DNS i przypisany do niej certyfikat klucza publicznego, w naszym przypadku przy uruchomieniu systemu w infrastrukturze AWS kwestie te pominięto i używany jest protokół HTTP. Serwer sam w sobie nie obsługuje protokołu HTTPS, w tym celu można wykorzystać reverse proxy takie jak Nginx, które będzie kończyć sesje TLS i przekazywać żądania klienta za pomocą protokołu HTTP do Serwera. Komunikacja między Serwerem, a Nginx musi być również chroniona, w przypadku uruchomienia serwisu na infrastrukturze Google Cloud lub AWS będzie ona automatycznie szyfrowana na niższych warstwach OSI. Przy takiej architekturze Serwer powinien być dostępny z sieci wewnętrznej.

3.7.2. Obsługa plików

Pliki przechowywane są w systemie Google Cloud Storage. Po stronie *Serwera* w tabeli **Remote Files** zapisywane są jedynie metadane plików, w szczególności ścieżka pliku w kubełku, rozmiar pliku oraz identyfikator właściciela. Kubełek (ang. bucket) to abstrakcja wykorzystywana w wielu chmurowych systemach przechowywania plików, dlatego też możliwa byłaby dosyć prosta zmiana dostawcy usług chmurowych wybranego do tego celu. Każdy plik ma unikalną ścieżkę, kubełek zezwala na odczyt oraz zapis pliku adresowanego ścieżką. Nie jest możliwa operacja edycji pliku, trzeba wykonać usunięcie i ponowny zapis, biorąc pod uwagę wymagania naszego systemu nie jest to jednak problemem, ponieważ pliki są tylko tworzone, pobierane i usuwane. Pliki są automatycznie szyfrowane przez Google Cloud Storage, co minimalizuje zagrożenie wycieku prywatnych danych użytkowników.

Przesyłanie plików jest wykonywane bezpośrednio między urządzeniem przesyłającym, a chmurą, dzięki czemu łączna *Serwera* nie są obciążane dużą ilością niepotrzebnego transferu. Ma to szczególną zaletę w przypadku pobierania oraz przesyłania plików z maszyn wirtualnych uczestników zajęć - jeśli tylko kubełek oraz instancje maszyn są fizycznie zlokalizowane w tym samym regionie to transfer jest bardzo szybki oraz w ogromnym stopniu odporny na awarie sieciowe. W związku z tym zalecana jest konfiguracja systemu, w której obie platformy chmurowe są w tym samym regionie, domyślnie jest to europe-central2, z centrami danych położonymi w okolicach Warszawy.

Na rysunku 30 przedstawiono diagram sekwencji opisujący przesłanie pliku od klienta, którym może być zarówno *Aplikacja Webowa*, jak i maszyna wirtualna, na której uruchomiono moduł.



Rysunek 30: Diagram sekwencji przesyłania pliku

Protokół może lekko przypominać three-way handshake z protokołu TCP i działa następująco:

1. Klient wysyła do *Serwera* prośbę o dokonanie transferu pliku, dołącza rozmiar pliku, identyfikator użytkownika oraz informacje o kontekście pliku (np. czy to plik z wynikami prac, czy zwykły plik z instrukcją nauczyciela).
2. *Serwer* weryfikuje czy użytkownik o tym identyfikatorze ma prawo przesyłać plik w danym kontekście (przykładowo uczeń nie może wysłać pliku z instrukcją od nauczyciela), sprawdzane jest też czy rozmiar pliku nie jest zbyt duży.
3. *Serwer* generuje dane żądania HTTP, takie jak adres URL, metoda HTTP i nagłówki, następnie podpisuje je cyfrowo i wysyła do klienta. Na rysunku 31 przedstawiono przykładowe zwracane dane w formacie yaml. Pole *signedUrl* zawiera dane pozwalające zweryfikować pozwolenie przesłania pliku po stronie serwisu chmurowego.

```
provider: GCLOUD
filePath: courses/1/activities/1/testMedium.dat
signedUrl: https://storage.googleapis.com/
swozo-dev-bucket-with-at-most-64-chars-kntp1/courses/1/activities/1/
testMedium.dat?
X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=...&
X-Goog-Date=data-podispania&X-Goog-Expires=czas-ważności&
X-Goog-SignedHeaders=podpisane-nagłówki-http&
X-Goog-Signature=podpis-cyfrowy-serwera
validTo: '2022-11-12T21:43:40.378Z'
httpMethod: PUT
httpHeaders:
  X-Goog-Content-Length-Range: 0,25165824
```

Rysunek 31: Dane umożliwiające dokonanie transferu pliku za pomocą protokołu HTTP

4. Klient generuje zapytanie HTTP wykorzystując podane informacje oraz wysyła je, dołączając plik, który ma zostać przesłany.
5. Serwis chmurowy weryfikuje podpis cyfrowy *Serwera*, sprawdza czy przesyłany plik ma rzeczywiście wskazany początkowo przez Klienta rozmiar i czy data ważności pozwolenia nie wygasła. Następnie zapisuje plik w kubełku.
6. Klient wysyła potwierdzenie przesłania pliku do *Serwera*, dołączając te same dane, które wcześniej wysłał mu *Serwer*.
7. *Serwer* weryfikuje podpis cyfrowy i zapisuje metadane pliku w bazie danych.
8. *Klient* oraz upoważnieni użytkownicy (w zależności od kontekstu) mogą operować na pliku w systemie, na przykład go pobrać.

Kroki w punkcie 4 pozwalają na obsługę przesyłu plików niezależną od dostawcy usług chmurowych, jedyne co musi zrobić Klient to wysłać przygotowane przez *Serwer* zapytanie HTTP. Podpis cyfrowy umożliwia weryfikację, czy dany użytkownik ma prawo przesyłać dany

plik, ma ograniczony czas ważności żeby to prawo nie było wieczne, ponieważ warunki mogą się zmienić (na przykład administrator może ograniczyć maksymalny rozmiar danych możliwych do przesłania przez użytkownika). Krok 6 jest konieczny, ponieważ wcześniej system nie zapisuje żadnych informacji o tym pliku, bo jeszcze nie istnieje. Należy zaznaczyć, że w przypadku awarii komunikacji sieciowej może zdarzyć się sytuacja, że przesłanie pliku do serwisu chmurowego powiedzie się, a nie uda się tego przesyłu potwierdzić *Serwerowi*, wówczas taki plik zajmuje miejsce w serwisie chmurowym, a nie da się go odczytać w systemie. Z tego powodu co 24 godziny uruchamiany jest CRON job usuwający takie niepotwierdzone pliki. W punkcie 7 jest potrzebna kolejna weryfikacja podpisu, żeby uchronić się przed spoofingiem, tzn. atakujący mógłby wysłać same potwierdzenia przesłania plików, które nie istnieją i spowodować niespójność w bazie danych.

Obsługa pobierania plików jest bardzo podobna i obejmuje kroki 1-5, Klient musi początkowo podać identyfikator pliku oraz obsłużyć jego pobieranie. Rysunek 32 przedstawia interfejs służący do obsługi plików, zaimplementowano wersję dla platformy Google Cloud, jednak takie same mechanizmy są obsługiwane chociażby przez AWS. Dane zaprezentowane na rysunku 31 odpowiadają danym zwróconym przez funkcję *createAuthorizedUploadRequest* pokazaną na rysunku 32.

```
public interface StorageProvider {
    StorageAccessRequest createAuthorizedUploadRequest(
        String bucketName, String storageObjectName, long maxFileSizeBytes, Duration validity);

    StorageAccessRequest createAuthorizedDownloadRequest(
        String bucketName, String storageObjectName, Duration validity);

    boolean isValid(StorageAccessRequest storageAccessRequest);

    CloudProvider getProviderType();

    @Async
    CompletableFuture<Void> cleanup(String bucketName, String storageObjectName);
}
```

Rysunek 32: Interfejs służący do obsługi plików po stronie *Serwera*

3.7.3. Komunikacja między *Serwerem*, a *Orkiestratorem*

Serwer do komunikacji z *Orkiestratorem* wykorzystuje REST API. Jest to komunikacja dwustronna i żaden z podsystemów nie pełni jedynie roli klienta bądź serwera.

Po stronie *Serwera* wydzielone są dedykowane endpointy dla *Orkiestratora*, służą one między innymi pobieraniu danych o uczestnikach zajęć, otrzymywaniu zezwolenia na transfer plików, a także ustawianiu linków do zajęć po uruchomieniu przez *Orkiestrator* modułów. Są to endpointy wewnętrzne, dlatego konieczne jest ich odpowiednie zabezpieczenie, przykładowo nie można dopuścić, żeby atakujący z zewnątrz mógł ustawić linki do niebezpiecznych stron dla zwykłych użytkowników systemu. W tym celu zastosowano uwierzytelnianie za pomocą współdzielonego klucza tajnego. *Orkiestrator* w każdym zapytaniu HTTP do *Serwera* ustawia nagłówek *Authorization*, w którym przesyła klucz. *Serwer* po otrzymaniu żądania sprawdza czy zgadza się on z kluczem przechowywanym lokalnie, jeśli nie to odrzuca żądanie. Symetryczny mechanizm po stronie *Orkiestratora* nie jest konieczny, ponieważ ustawiono zaporę sieciową (Firewall) umożliwiającą komunikację HTTP do *Orkiestratora* tylko z *Serwera*. Kluczowe jest, żeby niepowołane strony nie miały dostępu do przesyłanego klucza, dlatego zapytanie HTTP musi być szyfrowane. Jak już wspomniano w sekcji 3.7.1 w infrastrukturze Google Cloud dzieje się to automatycznie, w przeciwnym wypadku również komunikacji między tymi serwisami powinna być zabezpieczona, przykładowo tunelem IPsec. Klucz powinien być silnopseudolosowym ciągiem znaków długości minimum 256 bitów (32 znaki ASCII).

W celu usprawnienia komunikacji zaimplementowano algorytm exponential backoff, służący do powtarzania zapytań, których nie udało się wysłać z powodu chwilowej utraty połączenia. Algorytm polega na wysłaniu zapytania HTTP i w razie porażki oczekaniu pewnego czasu i wysłaniu go ponownie, z każdą próbą czas czekania jest zwiększany aż osiągnięty zostanie limit i zwrócony zostanie błąd do wysyłającego, który może go poprawnie w zależności od sytuacji obsłużyć. Liczba powtórzeń oraz czas czekania jest różny, przykładowo do wysłania linków przez *Orkiestrator* do *Serwera* wykonywane jest do 10 powtórzeń, ograniczając czas jednorazowego czekania do 3 minut. Jedną z sytuacji w których może się to przydać jest chwilowe przeciążenie *Serwera*, mechanizm automatycznego skalowania (ang. Auto Scaling) może uruchomić wówczas jego kolejną instancję. Bez oczekania przez *Orkiestrator* nie zdążyłaby się ona uruchomić i linki nie zostałyby dostarczone.

Jednym z często wysyłanych przez *Serwer* do *Orkiestratora* żądań jest zapytanie o dostępne serwisy i ich konfiguracje. Dane te nie powinny się zmieniać w trakcie działania systemu, a jednak często byłyby niepotrzebnie przesyłane, przykładowo za każdym razem kiedy nauczyciel chciałby utworzyć nowy moduł. Żeby zredukować czas odpowiedzi *Serwera* do *Aplikacji Webowej* oraz zmniejszyć obciążenie *Orkiestratora* zastosowano mechanizm pamięci podręcznej (cache). Jeśli konfiguracje serwisów były już raz otrzymane od *Orkiestratora* oraz czas, w którym to nastąpiło nie jest zbyt dawno to natychmiast zostaną zwrócone. Dzięki przedawnianiu pamięci cache resetowanie *Serwera* po dodaniu nowego serwisu nie będzie konieczne, wystarczy oczekać pewien czas (domyślnie 30 minut) i *Serwer* ponowi zapytanie o te dane do *Orkiestratora*. Zaimplementowano mechanizmy gwarantujące bezpieczne zachowanie pamięci cache w środowisku wielowątkowym, w tym przypadku było to bardzo proste, ponieważ dane te są tylko odczytywane.

3.7.4. Komunikacja między *Aplikacją Webową*, a *Serwerem*

Aplikacja Webowa komunikuje się z *Serwerem* za pomocą REST API. Jak już wspomniano w sekcji 3.2.1 do generacji stubów wykorzystano generator Open API. Żeby generator mógł wygenerować odpowiedni kod potrzebna jest specyfikacja Open API, do jej automatycznego

tworzenia wykorzystano bibliotekę Swagger. W języku Java polega to na dodawaniu adnotacji do funkcji obsługujących dane zapytania HTTP oraz do zwracanych typów danych.

Rysunek 33 przedstawia przykładową funkcję z dodanymi adnotacjami, rysunek 34 pokazuje wygenerowany na ich podstawie schemat Open API. Za pomocą adnotacji określono, że funkcja obsługuje żądanie POST, wysłane na endpoint `/policies`, dodatkowo konieczne jest użycie w nagłówku HTTP tokenu JWT. W ciele zapytania konieczne jest przesłanie danych zawierających między innymi pole numeryczne zawierające identyfikator nauczyciela oraz liczbę całkowitą z wartością polityki. Za pomocą adnotacji określono, że wszystkie te pola są wymagane.

```
@PostMapping(value = "/policies")
@PreAuthorize("hasRole('ADMIN')")
@SecurityRequirement(name = ACCESS_TOKEN)
public PolicyDto addPolicy(AccessToken token, @RequestBody CreatePolicyRequest createPolicyRequest) {
    logger.info("creating new policy for userId: {}", createPolicyRequest.teacherId());
    return policyService.createPolicy(createPolicyRequest);
}

public record CreatePolicyRequest(
    @Schema(required = true) PolicyType policyType,
    @Schema(required = true) Long teacherId,
    @Schema(required = true) Integer value
) {
```

Rysunek 33: Przykładowy endpoint odpowiedzialny za tworzenie polityki w systemie

```
/policies:
post:
  operationId: addPolicy
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/CreatePolicyRequest'
        required: true
  responses:
    "200":
      description: OK
      content:
        '*/*':
          schema:
            $ref: '#/components/schemas/PolicyDto'
  security:
    - JWT_AUTH: []
```

	<pre>CreatePolicyRequest: required: - policyType - teacherId - value type: object properties: // ... teacherId: type: integer format: int64 value: type: integer format: int32</pre>
--	--

Rysunek 34: Wygenerowana specyfikacja w standardzie Open API

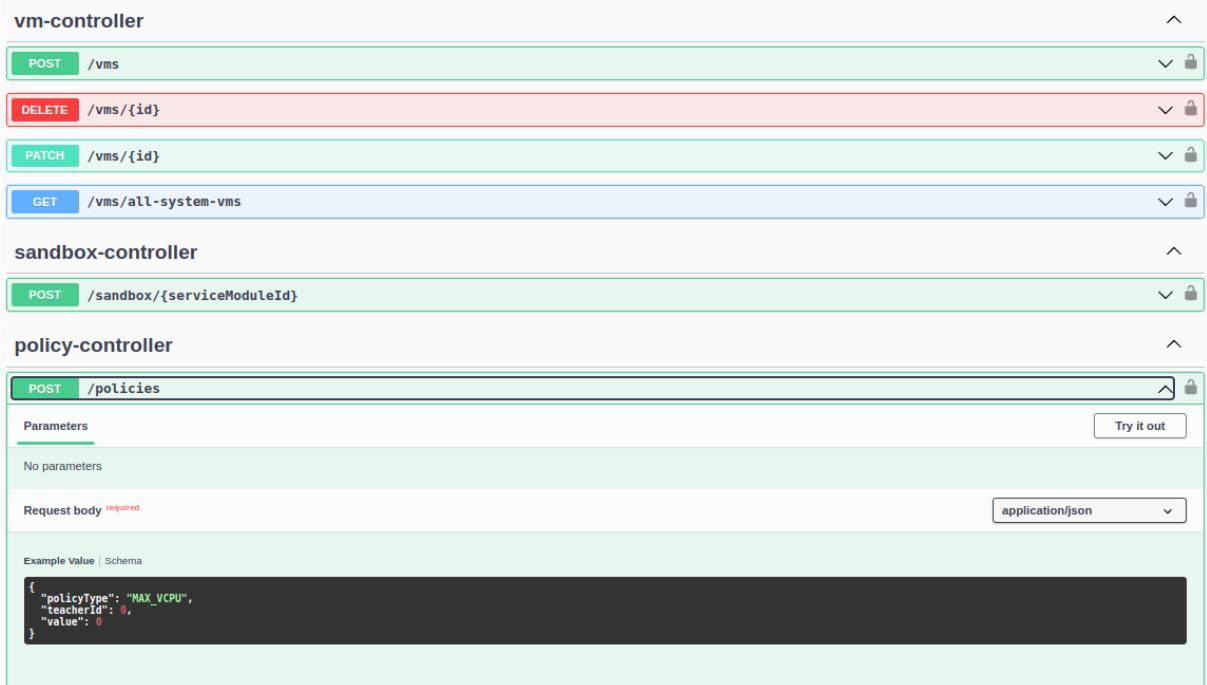
Specyfikacja Open API jest w pełni niezależna od technologii, w naszym przypadku generator miał za zadanie wygenerowanie kodu w języku TypeScript. Wygenerowany kod jest dosyć skomplikowany i zostanie tu pominięty, jednak samo jego wykorzystanie jest bardzo wygodne, na rysunku 35 pokazano wywołanie wygenerowanej funkcji w języku TypeScript, które samo w sobie jest bardzo czytelne i proste. Wykonanie tego kodu spowoduje wysłanie zgodnego ze schematem Open API zapytania HTTP do *Serwera*. Wszystkie typy odpowiadają tym, których oczekuje *Serwer*, co w ogromnym stopniu redukuje szansę popełnienia błędu.

```
apis.policyApi.addPolicy({
    createPolicyRequest: { policyType: CreatePolicyRequestPolicyTypeEnum.Vcpu, teacherId: 1, value: 1 },
});
```

Rysunek 35: Wywołanie wygenerowanej funkcji odpowiedzialnej za wysłanie zapytania HTTP

Dzięki wykorzystaniu wygenerowanego kodu można skupić się na implementowaniu logiki systemu, a nie na ręcznym przepisywaniu typów zmiennych oraz pamiętaniu jaka metoda HTTP musi być wysłana na jaki adres URL.

Kolejną zaletą jest automatyczne generowanie dokumentacji API, wystarczy otworzyć w przeglądarce stronę wygenerowaną przez bibliotekę Swagger i zostaną wyświetlane wszystkie obsługiwane przez *Serwer* endpointy HTTP, wraz z oczekiwany całkiem zapytań oraz zwracanymi wynikami. Rysunek 36 przedstawia część wygenerowanej strony.



Rysunek 36: Wygenerowana dokumentacja API

Wygenerowany kod nie zapewnia jednak potrzebnych mechanizmów QoS. Z tego powodu w *Aplikacji Webowej* zaimplementowano algorytm exponential backoff opisany w sekcji 3.7.3, w przypadku przekroczenia czasu powtórzeń zostanie zwrócony błąd do użytkownika informujący o utracie połączenia.

Dodatkowo z uwagi na wygodę użytkownika wykorzystano mechanizm zapisywania wyników zapytań w pamięci podręcznej (cache) klienta. Jeśli odpowiedź na wysyłane niedawno zapytanie znajduje się w pamięci cache to zostanie ona natychmiast zwrócona, pozwala to na przykład na bardzo szybkie wygenerowanie zawartości strony internetowej, dzięki czemu użytkownik nie widzi ładowania. W tle wysyłane jest zapytanie, którego wynik odświeży pamięć podręczną, dzięki czemu na stronie nie są wyświetlane przestarzałe dane. Do bezpiecznego zarządzania pamięcią cache wykorzystano bibliotekę React Query.

3.7.5. Obsługa maszyn wirtualnych

Obsługa maszyn wirtualnych znajduje się w podsystemie *Orkiestratora*. Cykl życia każdej z nich jest ściśle powiązany z cyklem życia serwisów na niej przygotowywanych, o którym można przeczytać w sekcji 3.7.6 Implementacja zarządzania maszynami wirtualnymi korzysta z interfejsu *TimedVmProvider*, przedstawionego na rysunku 37, co pozwoli na podmianę dostawcy maszyn wirtualnych bez ingerencji w resztę kodu.

```
public interface TimedVmProvider {  
    CompletableFuture<VmResourceDetails> createInstance(MdaVmSpecs mdaVmSpecs, String namePrefix);  
  
    CompletableFuture<VmResourceDetails> getVMResourceDetails(long internalResourceId);  
  
    CompletableFuture<Void> deleteInstance(long internalResourceId);  
  
    int getVMCreationTime(MdaVmSpecs mdaVmSpecs);  
}
```

Rysunek 37: Interfejs *TimedVmProvider* do obsługi cyklu życia maszyn wirtualnych

Obecnie jedyna implementacja interfejsu *TimedVmProvider* korzysta z usługi Compute Engine na platformie Google Cloud. Została do tego wykorzystana biblioteka *google-compute-engine* umożliwiająca programową interakcję z API Compute Engine. Rysunek 38 przedstawia fragment kodu odpowiedzialnego za konfigurację żądania utworzenia instancji maszyny wirtualnej.

```
private InsertInstanceRequest createInsertInstanceRequest(VmAddress vmAddress, VMSpecs vmSpecs) {  
    var instance = createInstanceRepresentation(vmAddress, vmSpecs);  
  
    return InsertInstanceRequest.newBuilder()  
        .setProject(vmAddress.project())  
        .setZone(vmAddress.zone())  
        .setInstanceResource(instance)  
        .build();  
}  
  
private Instance createInstanceRepresentation(VmAddress vmAddress, VMSpecs vmSpecs) {  
    var disk = diskProvider.createDisk(DEFAULT_DISK_NAME, vmSpecs.imageFamily(), vmSpecs.diskSizeGB());  
  
    var networkInterface = networkInterfaceProvider.createNetworkInterface(vmAddress.networkName());  
  
    return instanceProvider.createInstance(vmAddress, vmSpecs, disk, networkInterface);  
}
```

Rysunek 38: Proces tworzenia żądania utworzenia maszyny wirtualnej

3.7.6. Przepływ danych w *Orkiestratorze* i schemat zarządzania serwisami

Komunikacja pomiędzy *Orkiestratorem*, a *Serwerem* jest dwustronna i rozpoczyna się w momencie, gdy *Orkiestrator* przyjmie od *Serwera* żądanie przygotowania serwisów dla wybranych przez użytkownika aktywności. Każde takie żądanie ma przypisany sobie czas życia i listę serwisów, które należy przygotować w obrębie jednej instancji maszyny wirtualnej. Po otrzymaniu żądania *Orkiestrator* ma za zadanie przygotować wszystkie serwisy i dostarczyć *Serwerowi* informacje pozwalające na połączenie się użytkowników.

Orchestrator / Schedule Jupyter

The screenshot shows a POST request to `localhost:8080/schedules...`. The **Body** tab is selected, showing the following JSON payload:

```

1  {
2    "serviceLifespan": {
3      "startTime": "2022-11-12T22:20:47.4281697",
4      "endTime": "2022-11-13T22:57:47.4281697"
5    },
6    "mdaVmSpecs": {
7      "machineType": "e2-medium",
8      "diskSizeGb": 10
9    },
10   "serviceDescriptions": [
11     {
12       "activityModuleId": 1,
13       "serviceType": "JUPYTER",
14       "dynamicProperties": {
15         "notebookLocation": "1"
16       }
17     }
18   ]
19 }

```

Rysunek 39: Żądanie utworzenia instancji z jednym serwisem Jupyter Notebook

Schemat przygotowywania każdego serwisu można opisać w postaci pięciu głównych zadań:

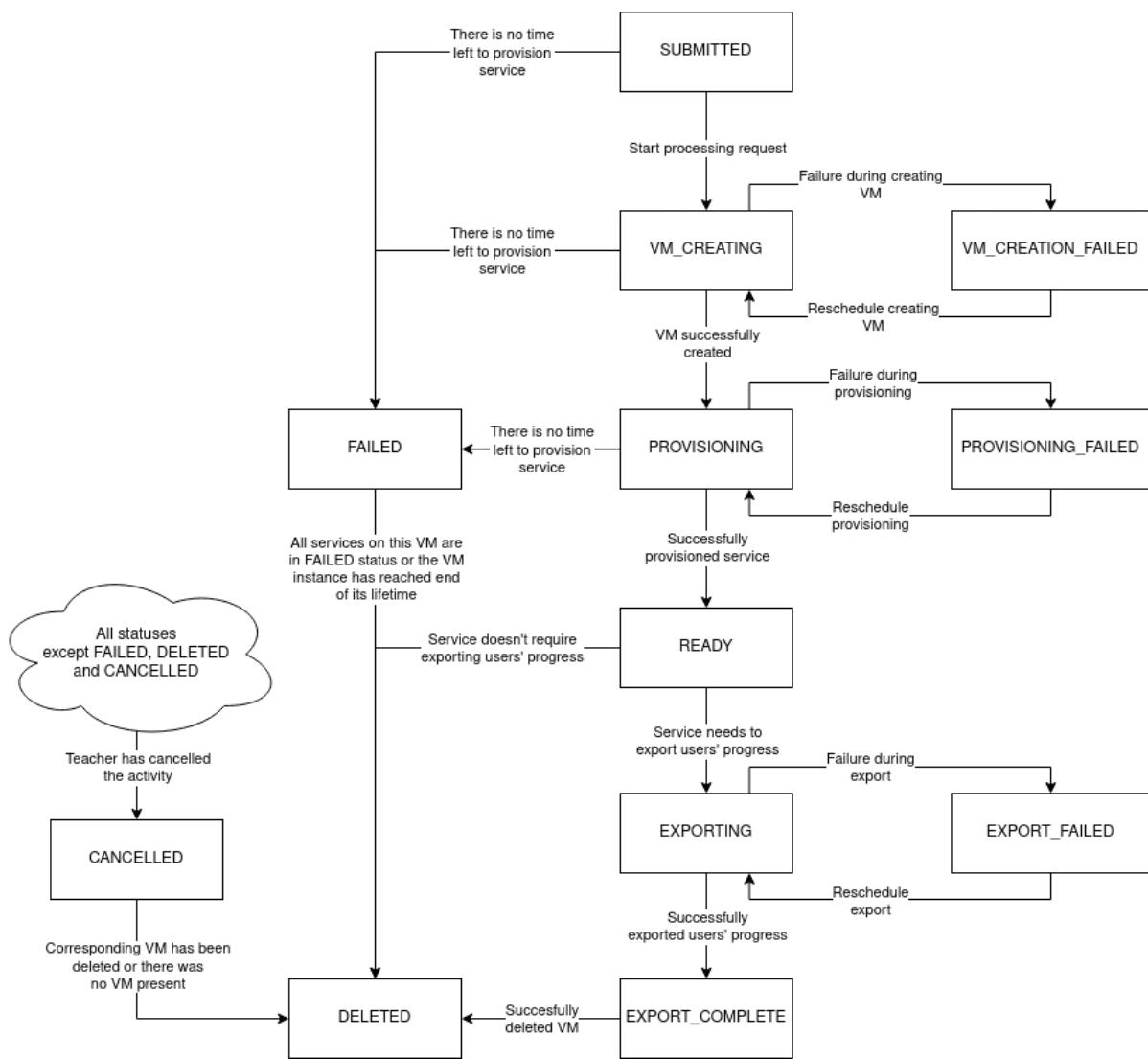
- Utworzenie maszyny wirtualnej o specyfikacji wyliczonej przez *Silnik MDA*;
- Przygotowanie na maszynie serwisów wyspecyfikowanych w żądaniu od *Serwera*;
- Dostarczenie do *Serwera* instrukcji podłączenia się do serwisu wraz z linkami umożliwiającymi podłączenie;
- Wyeksportowanie prac użytkowników do zewnętrznego serwisu przechowującego pliki, jeśli dany serwis tego wymaga;
- Usunięcie maszyny wirtualnej, gdy czas życia serwisu dobiegnie końca, a eksport plików się powiedzie.

3.7.7. Motywacja i zasady działania mechanizmów niezawodności w *Orkiestratorze*

Podczas obsługi każdego żądania przez *Orkiestrator* mogą wystąpić problemy, których ryzyka wystąpienia nie da się wyeliminować. Wpływ na to mogą mieć między innymi awarie

sieci, niedostępność serwisów dostawcy chmurowego, niedostępność naszych aplikacji, czy nawet przypadkowe wyłączenie *Orkiestratora*. Takie warunki wskazują na konieczność zaimplementowania mechanizmów niezawodności i odzyskiwania poprawnego stanu w przypadku niepowodzenia przetwarzanych żądań. Aby to umożliwić wykorzystana została baza danych, w której rejestrowane są stany wszystkich istotnych zasobów. Zakładając, że przygotowywanie jakiegoś serwisu nie powiedzie się na pewnym etapie, będzie to jedynie chwilowy problem. Dzięki śledzeniu statusów mogliśmy wdrożyć cykliczne weryfikowanie i wznowianie żądań, które nie dobiegły końca. W przypadku restartu całego systemu obecne konfiguracje zaczytywane są z bazy danych i wznowiane tak, jak w przypadku niepowodzeń - od pierwszego etapu, który nie był w stanie się wykonać.

Poniżej przedstawiony został diagram maszyny stanowej reprezentujący cykl życia każdego serwisu obsługiwanego przez *Orkiestrator*.



Rysunek 40: Maszyna stanów cyklu życia serwisów w *Orkiestratorze*

Po zaakceptowaniu żądania od *Serwera*, *Orkiestrator* umieszcza każdy z serwisów wymienionych w żądaniu w stanie SUBMITTED. Następnie rozpoczyna się proces ich przygotowania i usuwania. Stany wymienione na diagramie są od siebie zależne, aby przejść do następnego poprawnego stanu, konieczne jest pomyślne wykonanie zadań należących do poprzednich stanów.

Trzy główne stany, w których można spodziewać się porażki to tworzenie instancji maszyny wirtualnej (VM_CREATING), uruchamianie serwisu (PROVISIONING) i eksport plików do zewnętrznego magazynu danych (EXPORTING). Każdy z nich ma odpowiadający sobie stan z sufiksem "_FAILED" reprezentujący porażkę na tym etapie. Proces przygotowania serwisów przebywających w tych nieprawidłowych stanach jest cyklicznie wznowiany, co z powrotem przenosi je do stanu bez sufiksu. Obsługę danego żądania od *Serwera* można uznać za skończoną, w momencie gdy wszystkie serwisy w nim zawarte trafią do stanu DELETED, reprezentującego zwolnienie przez nie zasobów. Do stanu DELETED serwis może trafić po udanym obsłużeniu serwisu (stany READY i EXPORT_COMPLETE), po jego anulowaniu (CANCELLED), a także po definitivej porażce w trakcie obsługi serwisu (FAILED).

3.7.8. Integracja nowych serwisów

Ze względu na fakt, że jednym z zadań *Orkiestratora* jest przygotowywanie do działania udostępnianych użytkownikom serwisów, to właśnie on będzie podsystemem wymagającym rozszerzania, wraz ze wzrostem liczby zintegrowanych serwisów. Projektując jego działanie staraliśmy się umożliwić jego rozbudowywanie jak najmniejszym kosztem. Dopóki wymagania biznesowe odnośnie tego podsystemu nie rozszerzą się, integracja nowego serwisu powinna wymagać jedynie zaimplementowania nowej klasy przygotowującej dany serwis na maszynie wirtualnej. Niezawodność systemu została zapewniona niezależnie od typu przygotowywanej aplikacji.

Rysunek 41 przedstawia interfejs, który należy zaimplementować dodając wsparcie dla nowego serwisu. Główną metodą odpowiedzialną za właściwą konfigurację serwisu jest metoda *provision*, przyjmuje ona między innymi parametry wprowadzone przez nauczyciela technicznego oraz informacje pozwalające na ustalenie połączenia z uprzednio uruchomioną maszyną wirtualną. Metoda zwraca obietnicę (ang. promise) zawierającą listę linków do serwisu, które po spełnieniu obietnicy zostaną wysłane do *Serwera*. Jeśli serwis obsługuje opcję zapisu plików z zajęć to metoda *getWorkdirToSave* musi zwracać ścieżkę do folderu z plikami.

```
public interface TimedSoftwareProvisioner {
    CompletableFuture<List<ActivityLinkInfo>> provision(
        ScheduleRequestEntity requestEntity,
        ServiceDescriptionEntity description,
        VmResourceDetails resourceDetails
    );
    void validateParameters(Map<String, String> dynamicParameters) throws InvalidParametersException;
    ServiceTypeEntity getServiceType();
    ServiceConfig getServiceConfig();
    int getProvisioningSeconds();
    Optional<String> getWorkdirToSave();
}
```

Rysunek 41: Interfejs *TimedSoftwareProvisioner* konieczny do zaimplementowania przy integracji każdego nowego serwisu

W sekcji 3.6.3 została opisana motywacja użycia dynamicznych parametrów przy tworzeniu modułów lekcyjnych, poniżej zostanie przedstawiony dokładny sposób ich działania.

Integrując nowy serwis konieczne jest zaimplementowanie metody zwracającej wymagania konfiguracyjne serwisu, jest to metoda `getServiceConfig` widoczna na rysunku 41. **ServiceConfig** zawiera wszystkie potrzebne informacje służące do wygenerowanie i obsłużenia dynamicznego formularza w *Aplikacji Webowej*. Rysunek 42 przedstawia przykładowy **ServiceConfig** dla serwisu QuizApp w formacie YAML. **ServiceConfig** zawiera również instrukcje konfiguracji serwisu, wraz z minimalnymi wymaganiami sprzętowymi, a także informacje o przeznaczeniu serwisu w formacie HTML, dla zwięzości dane te zostały pominięte na rysunku 42.

```
serviceName: QUIZAPP
parameterDescriptions:
- name: questionsLocation
  required: true
  type: FILE
  translatedLabel:
    PL: Pytania dla uczniów
  clientValidationHelpers:
    allowedExtensions:
      - yaml
      - yml
- name: quizDurationSeconds
  required: true
  type: TEXT
  translatedLabel:
    PL: Czas trwania quzu w sekundach
  clientValidationHelpers:
isolationModes:
- SHARED
```

Rysunek 42: **ServiceConfig** zwracany przez klasę obsługującą integrację serwisu QuizApp

Pole z nazwą serwisu (**serviceName**) umożliwia przydzielenie zadania konfiguracji danego serwisu właściwej klasie implementującej jego integrację po stronie *Orkiestratora*. Pole z możliwymi trybami izolacji (**isolationModes**) umożliwia ograniczenie wyboru konfiguracji specyfikacji MDA modułu tylko do dostępnych trybów pracy serwisu. Najważniejszym polem jest pole z opisem parametrów konfiguracyjnych (**parameterDescriptions**), każdy parametr zawiera nazwę, umożliwiającą późniejsze odnoszenie się do zakodowanych wartości wprowadzonych przez twórcę modułu, przykładowe wartości są widoczne na rysunku 43. Dodatkowo zdefiniowane są następujące pola:

- **required** - czy nauczyciel techniczny musi uzupełnić dane pole;
- **type** - jeden z możliwych typów pól, dynamiczna obsługa każdego parametru bazuje na jego typie, przykładowo dla tekstu nie trzeba nic robić, a dla pliku konieczna jest obsługa jego przesłania oraz zakodowanie identyfikatora jako pola tekstowego;
- **translatedLabel** - opis danego pola, z tłumaczeniami w obsługiwanych językach, ma zastosowanie tylko z punktu widzenia człowieka tworzącego moduł;

- **clientValidationHelpers** - obiekty umożliwiające walidację parametru, na przykład dla pliku można wymagać specyficznych rozszerzeń.

```
dynamicParameters:  
| questionsLocation: 463  
| quizDurationSeconds: 120
```

Rysunek 43: Przykładowe wartości dynamicznych parametrów dla serwisu QuizApp

Dla każdego serwisu można w dowolny sposób tworzyć listę wymaganych parametrów, każdy z nich jest w sposób generyczny i zależny od typu pola obsługiwany po stronie *Serwera* oraz *Aplikacji Webowej*. Dodanie nowych typów pól wymagałoby zaimplementowania ich obsługi po stronie obydwu wymienionych podsystemów, jednak dodanie nowego serwisu, jak wcześniej wspomniano, wymaga jedynie implementacji interfejsu przedstawionego na rysunku 41. Dla naszych zastosowań wystarczająca była implementacja generycznego wsparcia dla pól tekstowych oraz plików, w przypadku liczb wciąż można użyć pól tekstowych.

Wartości dynamicznych parametrów są przechowywane w bazie danych *Serwera* w tabeli **Service Module Dynamic Properties** jako mapowanie *nazwa parametru* → *wartość*. Wartość musi być zakodowana w postaci tekstu, dlatego w przypadku plików zapisywany jest jego identyfikator z tabeli **Remote Files**. *Orkiestrator* na podstawie typu pola jest w stanie stwierdzić, że zakodowana wartość jest identyfikatorem pliku i wie, że musi wysłać zapytanie do *Serwera* w celu jego pobrania.

W trakcie tworzenia modułu *Aplikacja Webowa* przesyła do *Serwera* dynamiczne parametry w postaci mapy *nazwa parametru* → *wartość*, wartość jest zakodowanym w formacie JSON obiektem, zależnym od typu pola (przykładowo dla pliku w fazie rezerwacji wysyłana jest zakodowana prośba o zezwolenie jego przesłania do kubelka, a w fazie potwierdzenia rezerwacji przesyłany jest zakodowany obiekt informujący o sukcesie przesyłu), *Serwer* jest odpowiedzialny za dekodowanie wartości i odpowiednią ich obsługę.

3.7.9. Przykład wykorzystania Ansible w celu integracji serwisu Jupyter

Interfejs TimedSoftwareVmProvider w żadnym stopniu nie obliguje deweloperów do wykorzystania Ansible w celu przygotowania integrowanego serwisu, ale jak już zostało napisane w sekcji 3.2.3 jest to narzędzie, które w znacznym stopniu upraszcza proces wdrażania aplikacji. Najczęściej jest ono używane przez administratora systemu z poziomu wiersza poleceń, ale wykorzystanie z poziomu kodu nie jest wykluczone. W klasie AnsibleRunner zaimplementowana została logika, która ułatwia programowe wykorzystanie Playbooków, a jej użycie jest sugerowanym sposobem integrowania nowych serwisów. Rysunek 44 przedstawia przykładowy Playbook wykorzystany w projekcie, służący do przekopiowania plików z lokalnej maszyny do zdalnego hosta.

```

---
- name: Create a directory if it does not exist
  import_playbook: ./create-directory.yml
  vars:
    path: '{{ path_to }}'

- name: Copy file to remote host
  hosts: all
  become: yes
  become_user: '{{ user }}'
  tasks:
    - name: Copy content of directory to remote host
      copy:
        src: '{{ path_from }}'
        dest: '{{ path_to }}'
        owner: '{{ user }}'
        group: '{{ user }}'
        mode: '0777'

```

Rysunek 44: Playbook służący do przekopiowania plików z lokalnej maszyny do zdalnego hosta

Playbook to osobny plik składający się z listy zadań (sekcja tasks), a każde zadanie wykorzystuje dostarczone przez Ansible lub społeczność moduły. Nazwy w podwójnych nawiasach klamrowych to zmienne, które umożliwiają modyfikowanie działanie Playbooka w momencie uruchomienia. Moduł można rozumieć jako pewnego rodzaju opakowanie na zbiór operacji w obrębie danego zasobu, przykładowo *ansible.builtin.file* dostarcza możliwość wykonania najczęstszych operacji na plikach, w przypadku zamieszczonego Playbooka - utworzenie katalogu i przekopiowanie do niego plików. Działanie każdego modułu można specyfikować poprzez nadawanie obsługiwany słowem kluczowym konkretnych wartości, a ich pełny opis najlepiej sprawdzać w dokumentacji Ansible [9].

Wszystkie wykorzystane Playbooki są umieszczone w repozytorium, wraz z kodem aplikacji. Dzięki funkcjonalności importowania, są one zdolne do wykorzystania w dalszych fazach rozwoju projektu. Na rysunku 45 zamieszczony został Playbook ilustrujący cały proces wdrożenia Jupytera. Główne kroki to kolejno: utworzenie użytkownika w systemie operacyjnym, zainstalowanie oprogramowania Docker, konfiguracja serwisu Jupyter i jego właściwe uruchomienie.

```
---
```

```
- name: Setup SWOZO user
  import_playbook: ../../administration/setup-user.yml
  vars:
    new_user: swozo

- name: Install docker
  import_playbook: ./docker/install-docker.yml

- name: Create directory for Jupyter config
  import_playbook: ../../administration/create-directory.yml
  vars:
    path: /home/swozo/.jupyter
    user: swozo

- name: Create directory for user's files
  import_playbook: ../../administration/create-directory.yml
  vars:
    path: /home/swozo/jupyter
    user: swozo

- name: Copy Jupyter config file
  hosts: all
  tasks:
    - name: Copy template and set password
      template:
        src: ./jupyter_notebook_config.j2
        dest: /home/swozo/.jupyter/jupyter_notebook_config.py
        owner: swozo

- name: Run jupyter
  import_playbook: run-jupyter.yml
```

Rysunek 45: Playbook odpowiedzialny za proces uruchomienia Jupytera

3.7.10. Obsługa wielu języków

Zaimplementowano obsługę języka polskiego oraz angielskiego. Każdy wykorzystany w kodzie *Aplikacji Webowej* fragment wyświetlanego tekstu jest kluczem do obiektu zawierającego wszystkie tłumaczenia w aktualnie wykorzystywanym języku. Implementacja obsługi kolejnego języka wymaga dodania nowego pliku z tłumaczeniami. Rysunek 46 przedstawia fragment pliku z tłumaczeniami w języku polskim. W podobny sposób należy dostarczyć tłumaczenia dla instrukcji obsługi zintegrowanych serwisów. Po stronie *Aplikacji Webowej* wykorzystano w tym celu bibliotekę I18n, po stronie back-endu zaimplementowano uproszczony odpowiednik tej biblioteki w języku Java.

```
"login": {
    "header": "Logowanie",
    "email": "Adres Email",
    "password": "Hasło",
    "loginButton": "Zaloguj się",
    "forgotPassword": "Nie pamiętasz hasła?",
    "invalidCredentials": "Niepoprawny email lub hasło",
    "forgot": {
        "header": "Resetowanie hasła",
    }
}
```

Rysunek 46: Fragment pliku w formacie JSON z tłumaczeniami w języku polskim

3.8. Administracja systemem

Z założenia wytworzony przez nas produkt ma być wykorzystywany w placówkach takich jak uczelnie czy firmy. Każde z tych miejsc ma mieć własną, niezależną kopię systemu, przeznaczoną jedynie dla swoich pracowników bądź uczniów. Z tego powodu konieczne jest opisanie procesu wdrażania systemu od podstaw, za samo wdrożenie, późniejsze utrzymanie i zarządzanie systemem będzie odpowiadał administrator danej placówki.

W ramach pracy inżynierskiej wdrożono i przetestowano działanie systemu w infrastrukturze AWS, z uwagi na ograniczenia darmowego konta nie wykorzystano wszystkich zalecanych tutaj mechanizmów.

Poniższa lista przedstawia wytyczne dotyczące wdrażania systemu, elementy opcjonalne zostaną wyraźnie zaznaczone.

1. Uzyskanie dostępu do kopii kodu źródłowego projektu.

Zaleca się skorzystanie z oficjalnej dystrybucji kodu źródłowego z platformy GitHub [12]. Najnowsza działająca i przetestowana wersja jest dostępna na głównej gałęzi projektu (ang. main branch).

2. Założenie konta użytkownika na platformie Google Cloud.

Należy utworzyć klucz umożliwiający dostęp do usług Google Cloud z poziomu oprogramowania (ang. service key¹⁵). Zaleca się wybór regionu jak najbliższego geograficznie do placówki, w której działa system.

¹⁵Instrukcja tworzenia klucza do usług Google Cloud: <https://cloud.google.com/iam/docs/creating-managing-service-account-keys>

3. Uruchomienie wszystkich podsystemów.

Dla każdego podsystemu w odpowiednich folderach projektu znajdują się pliki *Dockerfile* umożliwiające uruchomienie ich w formie kontenerów Dockera. Dokładne i aktualne wytyczne dotyczące ustawień zmiennych środowiskowych, otwartych portów i zamontowanych katalogów dostępne są w plikach *Readme* w repozytorium. Pliki *Readme* zawierają również opis wszystkich komend, które musi wprowadzić administrator celem uruchomienia projektu. Przygotowano również playbook Ansible, umożliwiający uruchomienie systemu (po wcześniejszej konfiguracji infrastruktury) za pomocą pojedynczej komendy.

System sam w sobie nie musi działać na platformie Google Cloud, choć jest to zalecane z uwagi na zwiększoną wydajność i niezawodność *Orkiestratora*, a także samą prostotę wdrażania oprogramowania.

4. Zapewnienie potrzebnych mechanizmów bezpieczeństwa.

Jednym dostępnym z sieci publicznej podsystemem powinien być *Serwer*, który jak wspomniano w sekcji 3.7.1 nie obsługuje protokołu HTTPS. Z tego powodu konieczne jest użycie reverse proxy, w przypadku infrastruktury Google Cloud można wykorzystać HTTPS Load Balancing, konfigurowalny ręcznie za pośrednictwem konsoli Google Cloud. W przeciwnym przypadku można uruchomić na przykład Nginx i skonfigurować go do tego zastosowania.

Konieczne jest ustawienie zapory sieciowej (ang. firewall) tak, by z sieci publicznej możliwy był jedynie ruch HTTPS do reverse proxy oraz do serwera dostarczającego pliki z kodem *Aplikacji Webowej*. W przypadku Google Cloud można wszystkie pliki umieścić w systemie Google Cloud Storage, w przeciwnym przypadku można wykorzystać to samo reverse proxy. Konieczna jest konfiguracja DNS'a tak, by system był dostępny za pośrednictwem czytelnych nazw, należy również zapewnić zaufany certyfikat klucza publicznego, żeby przeglądarki użytkowników nie blokowały możliwości otworzenia strony.

Ruch do *Orkiestratora* i baz danych musi być możliwy jedynie z sieci wewnętrznej. Jeśli komunikacja między wewnętrznymi podsystemami może przechodzić przez sieć publiczną to również musi być szyfrowana, można w tym celu użyć tunelowania, przykładowo IPsec. Jak już zaznaczono w sekcji 3.7.1, jeśli wszystkie podsystemy będą uruchomione w infrastrukturze Google Cloud to będzie to zapewnione automatycznie.

Należy skonfigurować wszystkie potrzebne klucze w zmiennych środowiskowych zgodnie z instrukcjami zamieszczonymi w *Readme*.

5. Zapewnienie odpowiedniej redundancji i skalowania zasobów systemu.

Ten krok jest w dużym stopniu zależny od wielkości placówki. Dokładne wytyczne nie zostaną podane, ponieważ nie zostały przeprowadzone potrzebne testy, w związku z tym konieczne jest początkowe ciągłe monitorowanie działania systemu i reagowanie w razie wystąpienia problemów takich jak przeciążenie.

Warto zapewnić replikację baz danych *Serwera* i *Orkiestratora*, zalecane jest również ustawienie automatycznego, okresowego tworzenia kopii zapasowych zapisanych danych (dotyczy to także plików zapisanych w systemie Google Cloud Storage). Dla zwiększenia niezawodności i dostępności systemu zaleca się również replikację i automatyczne skalowanie instancji *Serwera*, z uwagi na naturę działania systemu ruch w nocy z dużym prawdopodobieństwem będące minimalny, więc można oszczędzić koszty i ograniczyć liczbę instancji. W przypadku infrastruktury Google Cloud zaleca się replikację w różnych strefach dostępu (ang. availability zones).

6. Sprawdzenie łączności między wszystkimi podsystemami.

Serwer musi być osiągalny z *Aplikacją Webową*, należy poprawnie skonfigurować politykę CORS (Cross-origin Resource Sharing), ustawiając nazwę domeny *Aplikacji Webowej* (jeśli będzie różna od domeny *Serwera*). W konfiguracji *Aplikacji Webowej* trzeba również podać adres URL, pod którym dostępny jest *Serwer*. Bazy danych *Serwera* i *Orkiestratora* muszą być osiągalne z odpowiadających podsystemów. Należy również zapewnić dwustronną łączność między *Serwerem*, a *Orkiestratorem*.

7. Skonfigurowanie repozytorium maszyn wirtualnych.

Po uruchomieniu systemu dostępne jest jedynie konto administratora, z podanym w konfiguracji adresem e-mail. Należy zresetować hasło i zalogować się w *Aplikacji Webowej*. Konieczna jest konfiguracja możliwych do wykorzystania przez system maszyn wirtualnych, ich nazwy powinny odpowiadać wewnętrznym nazwom maszyn w systemie Google Cloud [8] (na przykład e2-medium), parametry techniczne również muszą być identyczne.

Po skonfigurowaniu tych danych należy przetestować zapis pliku oraz utworzenie maszyn wirtualnych w celu sprawdzenia, czy prawa dostępu do usług Google Cloud zostały poprawnie nadane. Można to zrobić poprzez utworzenie modułu i wybranie opcji utworzenia środowiska testowego, po udanym teście można usunąć moduł.

8. Utworzenie kont użytkowników i wysłanie im danych dostępowych.

Wykorzystując *Aplikację Webową* należy utworzyć konta użytkowników i przypisać im role w systemie, następnie używając zewnętrznych narzędzi należy dostarczyć im adresy e-mail i poinformować, że dostęp do systemu będzie możliwy po zmianie hasła. Możliwe (i w większych placówkach zalecane) jest utworzenie kolejnych kont administratorów w celu oddelegowania im części obowiązków, mogą oni mieć również dostęp do platformy Google Cloud.

9. Monitorowanie działania systemu.

Ten krok należy realizować przez cały okres działania systemu. Należy w szczególności zwrócić uwagę na koszty naliczane przez platformę Google Cloud¹⁶, jeśli byłyby one zbyt duże można skontaktować się z nauczycielami zużywającymi największe ilości zasobów, można również ograniczyć im tę możliwość poprzez zmiany wartości polityk w panelu administratora.

Konieczne jest stałe monitorowanie logów generowanych przez system i reagowanie w przypadku poziomu *ERROR*. Dodatkowo zaleca się monitorować stan plików i maszyn wirtualnych wykorzystując narzędzia udostępniane przez platformę Google Cloud.

3.9. Podsumowanie rozdziału

W rozdziale opisano najważniejsze aspekty związane z implementacją i wdrożeniem systemu. Uzasadniony został sposób spełnienia postawionych przed produktem wymagań, zostało pokazane jak system może być wykorzystany w rzeczywistych placówkach. W rozdziale 4 opisany zostanie proces, efektem którego powstał system opisany w tym rozdziale. Rozdział 5 ostatecznie podsumuje efekty prac, omówi dokładne wyniki oraz braki aktualnej implementacji i możliwości ich wypełnienia.

¹⁶Konsola Google Cloud: <https://console.cloud.google.com>

4. Organizacja pracy

Rozdział skupia się na udokumentowaniu procesu powstawania projektu, organizacji pracy w zespole oraz narzędzi używanych podczas całego procesu planowania i implementacji. Sekcja 4.1 ma na celu przedstawienie członków zespołu oraz pełnionych przez nich funkcji. Sekcja 4.2 skupia się na opisaniu przyjętej przez zespół metodyki pracy, na której oparto cały proces twórczy projektu. Sekcja 4.3 przybliża kolejne etapy realizacji projektu, wyjaśniając cel każdego z nich. Sekcja 4.4 listuje wszystkie najważniejsze wykorzystane w czasie pracy narzędzia oraz wyjaśnia sposób ich wykorzystania.

4.1. Osoby w projekcie

W skład osób biorących udział w powstawaniu pracy wchodził czteroosobowy zespół projektowy (Mikołaj Bul, Konrad Czerepak, Adam Niemiec, Szymon Stępień) oraz promotor pracy, dr inż. Sławomir Zieliński. Pomimo dużej ilości współpracujących ze sobą elementów systemu, każdy z członków zespołu brał czynny udział w planowaniu i implementacji każdego z nich. Wszyscy posiadali jednolite role zarówno w trakcie analizy technicznej, jak i programowania. Jesteśmy w stanie przydzielić osoby do poszczególnych podsystemów, przy których to powstawaniu dana osoba brała większy udział: implementacja części front-endowej aplikacji webowej i bezpieczeństwa systemu - Szymon Stępień, część back-endowa oraz moduł MDA - Konrad Czerepak, część back-endowa oraz integracja Sozisela - Adam Niemiec, orkiestrator oraz komunikacja z usługą chmurową - Mikołaj Bul.

4.2. Metodyka pracy

W celu zapewnienia konsekwentności oraz spójności w powstawaniu pracy wykorzystana została metoda Kanban, zapewniająca wizualizację całego procesu oraz umożliwiającą zarówno zarządzanie, jak i monitorowanie statusu pracy. Metoda wykorzystuje tablicę składającą się z kolumn definiujących stan poszczególnych zadań, np. "Do zrobienia", "W trakcie", "Wykonane". Każde z zadań początkowo umieszcza się w pierwszej z kolumn, a wraz z postępem pracy osoba przydzielona przemieszcza je do odpowiedniej kolumny. Do prowadzenia wirtualnej tablicy Kanban wykorzystano system Jira Software, natomiast same zadania przydzielano podczas regularnych spotkań całego zespołu.

Ze względu na występowanie wielu pomniejszych, współpracujących ze sobą serwisów składających się na całość systemu, by zapewnić optymalną współpracę wszystkich elementów konieczna była systematyczna komunikacja pomiędzy członkami zespołu. Dzięki narzędziu do prowadzeniu rozmów głosowych Discord organizowano spotkania, na których omawiano wykonaną pracę, rozdzielano zadania oraz podejmowano decyzje projektowe. Podczas rozmów tworzono notatki podsumowujące pomysły i wnioski, aby podjęte w trakcie spotkań decyzje zostały utrwalone. Co pewien czas konsultowano również postęp prac z promotorem, wtedy to prezentowano dotychczasowe osiągnięcia oraz określano dalszy kierunek rozbudowy systemu. W przypadku codziennej komunikacji do przekazywania informacji i umawiania spotkań służyła aplikacja Facebook Messenger.

4.3. Etapy realizacji projektu

Całość realizacji finalnego projektu była złożonym procesem, z którego jesteśmy w stanie wyodrębnić cztery główne etapy. Wykonanie poszczególnych faz pozwalało na rozpoczęcie następnej części prac.

- **Rozpoznanie w temacie projektu**

Przez rozpoczęciem realizacji właściwej części projektu, kluczowym było uprzednie wykonanie prac badawczych w kierunku obejmującym jego tematykę. W tym celu, na podstawie źródeł dostarczonych przez promotorą [2, 1, 14], zapoznano się ze wszystkimi niezbędnymi do zaplanowania prac pojęciami. Głównym zamysłem tego etapu było jak najdokładniejsze zrozumienie koncepcji Model Driven Architecture oraz orkiestracji mikroserwisów. Wspólna analiza i podsumowanie materiałów pozwoliła na usystematyzowanie uzyskanych informacji, a sporzązone następnie notatki pozwoliły na utrwalenie pozyskanej wiedzy. Z rozpoznaniem wiązało się również badanie rynku systemów do organizacji zajęć zdalnych, na podstawie którego powstałe wnioski pozwoliły na zrozumienie problemu, który niniejsza praca miała za cel rozwiązać. Dzięki powyższym działaniom uzyskano solidne podstawy teoretyczne pozwalające na przejście do etapu planowania.

- **Planowanie pracy**

Okres planowania skupiał się na jak najlepszym wykorzystaniu zgromadzonych informacji, w celu opracowania architektury całego systemu, koncentrując się zarówno na produkcie jako całości, jak i na poszczególnych współpracujących ze sobą komponentach. Każdy z planowanych elementów dokładnie omówiono, a następnie na podstawie wniosków sporządzono diagramy UML i schematy interfejsu użytkownika. Poprawność przygotowanych materiałów konsultowano z promotorem, a następnie umieszczone w pracy. Wykonano również analizę oprogramowania i narzędzi mających pomóc w realizacji projektu oraz wstępnie podzielono się zadaniami pomiędzy członkami zespołu. Z tak przygotowanym planem przystąpiono do etapu implementacji prototypu.

- **Implementacja prototypu**

Założeniem etapu tworzenia prototypu była implementacja działającego systemu, dostarczającego wybrane funkcjonalności. Pierwsza wersja stworzonego systemu obejmowała m.in. interfejs użytkownika pozwalający na planowanie kursów i zajęć, orkiestrator umożliwiający instancjonowanie pojedynczych maszyn wirtualnych dla poszczególnych użytkowników oraz integrację z serwisem Jupyter Notebook. Na podstawie prototypu przeanalizowano dokładnie proces jego implementacji, sporządzono wnioski oraz dalsze plany mające na celu stworzenie finalnej wersji produktu.

- **Implementacja finalnego produktu**

Ostatni etap projektu skupiał się na dokończeniu rozpoczętych już prac i dostarczeniu wszystkich zaplanowanych wcześniej funkcjonalności. Najważniejszymi z dodanych w końcowej części elementami były: moduł MDA, panel admina, integracja z Google Cloud Storage, pozwalająca na zarządzanie plikami z zajęć, możliwość tworzenia dzielonych maszyn wirtualnych oraz integracja serwisów Jitsi Meet i QuizApp. Podczas implementacji zespół regularnie konsultował się zarówno ze sobą, dyskutując na temat decyzji projektowych, jak i z promotorem, któremu zdawał raport ze stanu prac.

4.4. Wykorzystane praktyki i narzędzia

Pomimo pracy nad oddzielnymi częściami serwisu, by zagwarantować ich współpracę, wykorzystane zostało wiele narzędzi pozwalających na komunikację, sprawne tworzenie i kontrolowanie aktualnego stanu systemu oraz dostarczenie zaplanowanych funkcjonalności.

1. **Komunikacja** - Podstawowym środkiem komunikacji pomiędzy członkami zespołu była aplikacja Facebook Messenger. Dzięki niej, za pomocą wiadomości tekstowych, szybko przekazywano ważne informacje oraz planowano spotkania. Gdy tekstowa forma komunikacji nie wystarczała, a omawiana kwestia wymagała większego zaangażowania, umawiano się na rozmowy głosowe za pomocą aplikacji Discord. Rozmowy te służyły do dyskutowania ważniejszych decyzji implementacyjnych, planowania dalszych kroków oraz dzielenia się doświadczeniami i osiągnięciami z dotychczasowej pracy.

Do komunikacji z promotorem wykorzystywano aplikację Webex. Podczas spotkań konsultacyjnych zdawano raport z postępów prac nad projektem, omawiano dotychczas podjęte decyzje oraz planowano kolejne działania w celu rozwoju systemu.

2. **Organizacja pracy** - By konsekwentnie zarządzać poszczególnymi zadaniami przypisanymi do członków zespołu, wykorzystano oprogramowanie Jira Software. Udostępniona tam funkcja tablicy Kanban pozwoliła na realizację metodyki o tej samej nazwie oraz monitorowanie postępów prac, co znacznie usprawniło proces organizacji pracy całego zespołu. W trakci tworzenia tablicy przyjęto wersję 4-kolumnową z następującymi stanami przyjmowanymi przez zadania: "Do zrobienia", "W trakcie realizacji", "Kod do recenzji" i "Wykonane". Rysunek 47 przedstawia zrzut ekranu z tablicą zadań w końcowej fazie implementacji systemu.
3. **System kontroli wersji** - W celu utrzymania kontroli nad tworzonym kodem utworzone zostało repozytorium [12] w systemie GitHub. Dzięki temu każdy z członków zespołu mógł lokalnie pracować nad jego własną kopią całości kodu, nie martwiąc się o ingerowanie w pracę pozostałych osób. Pomimo trzymania całego projektu w jednej lokalizacji, każdy z elementów systemu przechowywany był w osobnym katalogu, co zapobiegło problemom z organizacją oraz konfiguracją kodu.

Po przydzieleniu zadania do konkretnej osoby, tworzyła ona w repozytorium nową gałąź, na której implementowała swoje rozwiązanie. Następnie w celu scalenia nowego kodu do głównej gałęzi projektu, musiał zostać stworzony tzw. *Pull Request*. Przyjętą przez zespół praktyką była konieczność pomyślnego przejścia recenzji implementacji każdego wykonanego zadania przed jego scaleniem. Pozostali członkowie wypisywali uwagi oraz sugestie pod poszczególnymi fragmentami kodu. Gdy reszta zespołu zatwierdziła zmiany, następowało scalenie gałęzi.

4. **Porządkowanie gromadzonych materiałów** - Wszystkie etapy realizacji projektu generowały duże ilości informacji, planów oraz notatek, które należało w rozsądny sposób przechowywać. Wykorzystując narzędzie Confluence, stworzone zostało wiki projektu, które systematycznie uzupełniano koniecznymi do zapamiętania danymi m.in. notatkami ze spotkań, czy linkami do źródeł i dokumentacji. Rysunek 48 pokazuje nazwy części notatek sporządzonych w trakcie pracy nad projektem.

Tablica Kanban

The screenshot shows a Jira Kanban board titled "Tablica Kanban". The board has four columns: "TO DO 1", "IN PROGRESS 4", "CODE REVIEW 2", and "DONE 7". Below the columns, there are two status messages: "Przyspiesz 1 zgłoszenie" and "Wszystkie pozostałe 13 zgłosz.". The board lists several tasks:

- Service instructions frontend**: Status: TO DO, assigned to AN, ID: SWOZO-48.
- Sozisal provisioner**: Status: IN PROGRESS, assigned to KC, ID: SWOZO-40.
- Admin support**: Status: IN PROGRESS, assigned to SS, ID: SWOZO-37.
- Refactoring Service Modules for MDA purpose**: Status: IN PROGRESS, assigned to KC, ID: SWOZO-45.
- Allow for cancelling requests in Orchestrator**: Status: IN PROGRESS, assigned to MB, ID: SWOZO-47.
- Quiz integration**: Status: IN PROGRESS, assigned to SS, ID: SWOZO-46.
- MDA-backend-side-preparation**: Status: IN PROGRESS, assigned to KC, ID: SWOZO-33.
- Integrate MDA engine into scheduling**: Status: IN PROGRESS, assigned to MB, ID: SWOZO-38.
- Provisioning multiple services on single VM**: Status: IN PROGRESS, assigned to MB, ID: SWOZO-39.

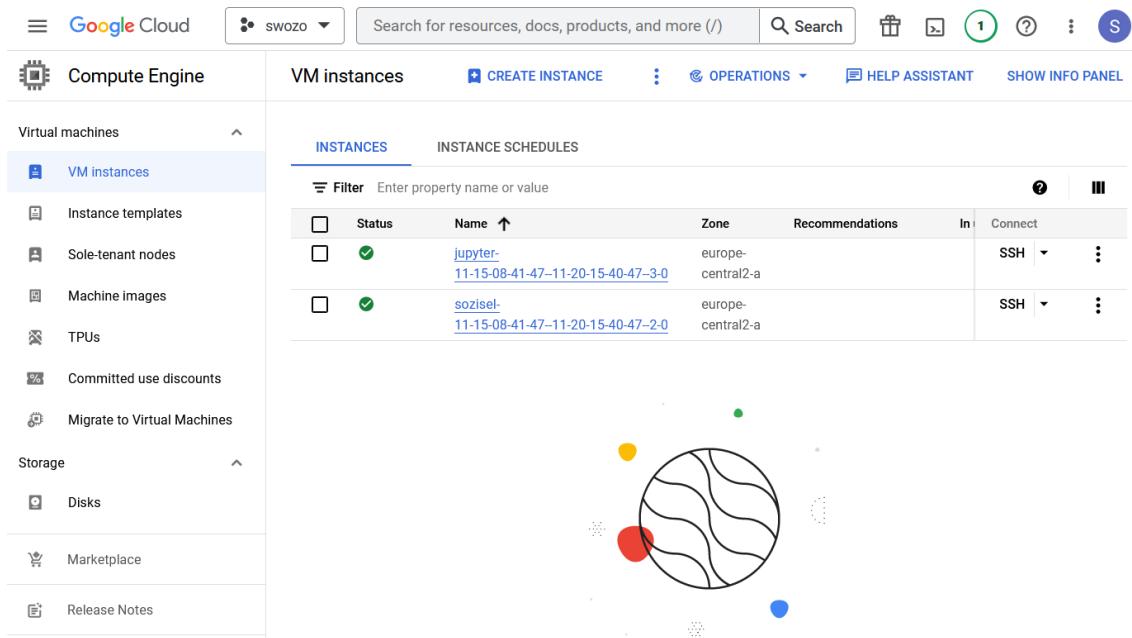
Rysunek 47: Tablica Kanban służąca do monitorowania postępów prac

The screenshot shows a Jira Software project page titled "Strony projektów". The sidebar on the left contains project navigation links: "System wspomagania ...", "Zgłoszenia", "Komponenty", "TWORZENIE OPROGRAMOWANIA", "Kod", "Wydania", "OPERACJE", "Wdrożenia", "Strony projektów" (selected), "SWOZO", "Dodaj skrót", and "Ustawienia projektu". The main area lists pages with their details:

Strona	Autor	Ostatnia aktualizacja
Dokumentacja	MB	Utworzono kwi 27. 2022
Analiza techniczna	MB	Utworzono kwi 27. 2022
Definicje wysokopoziomowe	MB	Zaktualizowano maj 07. 2022
Komponenty systemu	MB	Zaktualizowano kwi 29. 2022
Orkiestrator	MB	Zaktualizowano cze 06. 2022
Silnik MDA	MB KC	Zaktualizowano paź 20. 2022
Wybór chmury	MB	Zaktualizowano maj 14. 2022
Wybór technologii	MB	Utworzono maj 14. 2022
Sprawy organizacyjne	MB	Utworzono kwi 27. 2022
Plan konsultacji	MB	Zaktualizowano cze 13. 2022
Pytania z konsultacji	MB	Utworzono maj 14. 2022
Spotkania wewnętrzne	MB	Utworzono kwi 27. 2022
Spotkania z klientem	SS	Zaktualizowano maj 29. 2022

Rysunek 48: Wiki projektu stworzona w systemie Confluence

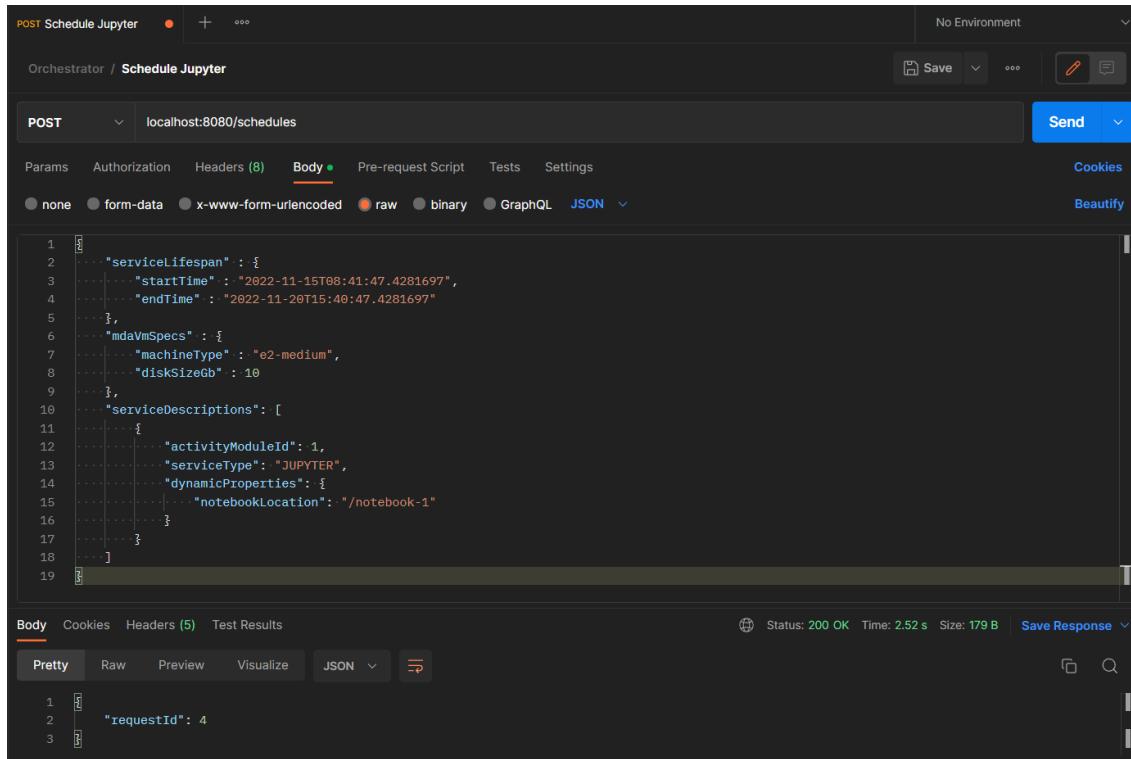
5. Google Cloud Platform - Opisany dokładnie w sekcji 3.1.4 wykorzystany serwis chmurowy Google to istotny element całego projektu. Ostateczna komunikacja systemu z chmurą zapewniona jest z poziomu kodu, lecz w trakcie implementacji oraz testowania korzystano z konsoli Google Cloud Platform, pozwalającej na manualne zarządzanie zasobami będącymi umiejscowionymi w chmurze. Rysunek 49 przedstawia panel zarządzania maszynami wirtualnymi wraz z uruchomionymi dwoma instancjami.



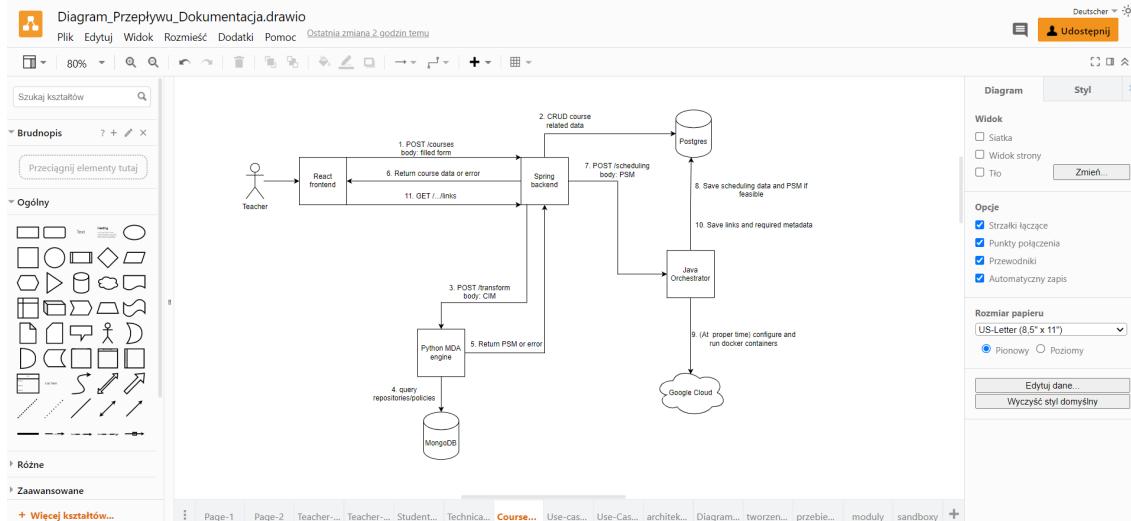
Rysunek 49: Konsola Google Cloud Platform - widok zarządzania maszynami wirtualnymi

6. Testowanie API - W celu testowania obsługi zapytań REST API oraz GraphQL wykorzystaliśmy oprogramowanie Postman - pozwalające na sprawne tworzenie i wysyłanie zapytań na odpowiedni punkt końcowy (ang. endpoint). Dzięki temu elementy systemu odbierające te zapytania mogły być testowane bez konieczności uruchamiania całości. W taki sposób zapytania wysyłano do *Serwera*, *Orkiestratora* oraz *Sozisela*. Rysunek 50 pokazuje interfejs aplikacji Postman, z widocznym wynikiem zapytania HTTP wysłanego do *Orkiestratora*.

7. Tworzenie diagramów - Do tworzenia diagramów, na podstawie których planowano architekturę systemu, a następnie umieszczonych w dokumentacji wykorzystano serwis *app.diagrams.net*. Jest to użyteczne narzędzie pozwalające na tworzenie zaawansowanych diagramów w języku UML. Wszystkie tworzone diagramy omawiano w gronie zespołu oraz konsultowano z promotorem, w celu potwierdzenia ich poprawności. Rysunek 51 przedstawia interfejs serwisu, na którym widoczny jest diagram z początkową wizją architektury systemu.

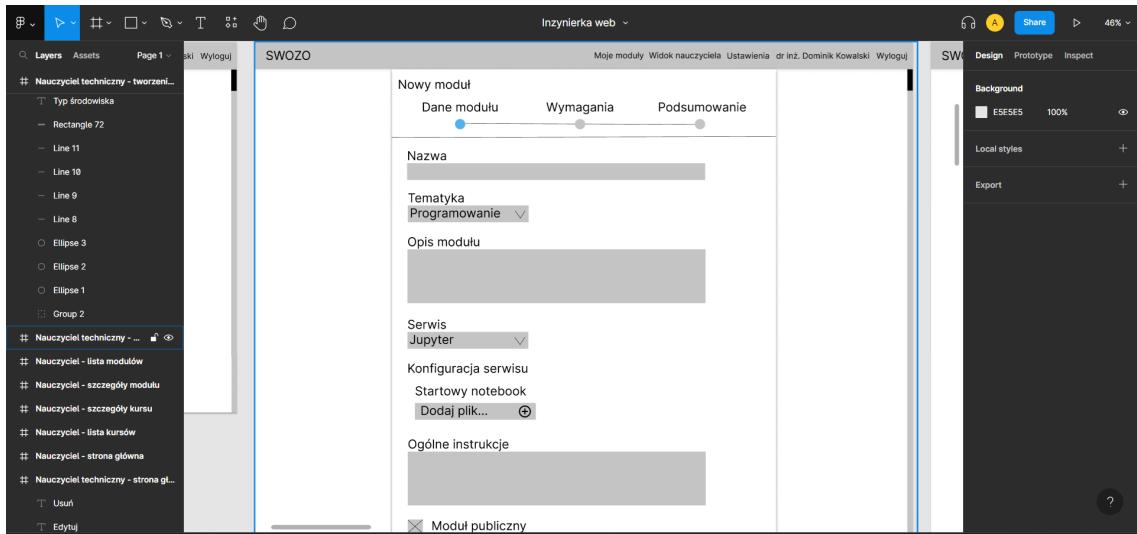


Rysunek 50: Interfejs programu Postman z przykładowym zapytaniem do planowania zajęć w *Orkiestratorze*



Rysunek 51: Interfejs serwisu *app.diagrams.net*

8. Projektowanie interfejsu użytkownika - Kolejnym wykorzystanym narzędziem do projektowania było oprogramowanie Figma, umożliwiające budowę szczegółowych makiet (ang. mockups) interfejsu użytkownika. Dzięki temu stworzono szkice panelu *Aplikacji Webowej* przed jej faktyczną implementacją. Rysunek 52 pokazuje interfejs strony Figma wraz z makietą tworzenia kursu, na podstawie której powstał odpowiadający widok w *Aplikacji Webowej*.



Rysunek 52: Interfejs serwisu Figma wraz z tworzenia kursu

4.5. Podsumowanie rozdziału

W rozdziale przedstawiono organizację całego procesu powstawania projektu. W głównej mierze skupiono się na ukazaniu systemu planowania i monitorowania postępów pracy, przedstawiono w nim poszczególnych członków zespołu, opisano metodykę oraz narzędzia wykorzystane podczas okresów planowania i implementacji. Nakreślono, w jaki sposób powstawał finalny produkt, z którego wnioski oraz wyniki zostaną przedstawione w kolejnym rozdziale.

5. Wyniki projektu

Rozdział podsumowuje efekty prac. Sekcja 5.1 prezentuje interfejs użytkownika i opisuje jego funkcjonalności. Sekcja 5.2 przybliża braki obecnej implementacji oraz wskazuje możliwe kierunki rozwoju systemu. Sekcja 5.3 przedstawia ocenę rezultatów pracy oraz omawia wnioski.

5.1. Prezentacja interfejsu użytkownika

Sekcja prezentuje interfejs użytkownika i opisuje jego funkcjonalności. Prezentacja obejmuje wszystkie grupy funkcjonalne opisane w sekcji 2.3. Dla lepszego wyjaśnienia działania systemu w sekcji 5.1.1 zaprezentowano przepływ sterowania opisany w sekcji 3.6, w sekcji 5.1.2 przedstawiono podsystem administracji systemem, a w sekcji 5.1.3 zebrano mniej powiązane widoki i opisano ich przeznaczenie.

5.1.1. Tworzenie modułów, kursów i przebieg aktywności edukacyjnych

W sekcji opisano pełny przepływ sterowania związanego z przygotowaniem przykładowego kursu z podstaw kryptografii. Przykład będzie obejmował tworzenie modułu bazującego na serwisie Jupyter Notebook, który następnie wraz z innymi utworzonymi w analogiczny sposób modułami zostanie wykorzystany w kursie. W przykładzie nauczyciel techniczny utworzy moduł, który potem sam wykorzysta w kursie, moduł zostanie udostępniony jako publiczny więc inni nauczyciele również będą mogli go wykorzystać we własnych kursach.

Rysunek 53 przedstawia stronę główną wyświetlaną po udanym zalogowaniu, pokazaną z punktu widzenia nauczyciela, który jest jednocześnie nauczycielem technicznym. W prawym górnym rogu znajdują się odnośniki umożliwiające nawigację do odpowiednich podstron, w przykładzie pokażemy podstronę z modułami oraz kursami. Strona główna zawiera kalendarz z planem zajęć, odnośniki do najbliższych aktywności oraz listę ostatnio przeprowadzonych aktywności.

The screenshot shows the main interface of the application. At the top, there is a blue header bar with the logo "SWOZO" on the left and navigation links "SERWISY", "PUBLICZNE MODUŁY", "MOJE MODUŁY", "MOJE KURSY", and a notification bell icon on the right.

The main content area is divided into several sections:

- Kalendarz (Calendar):** A monthly calendar for December 2022. Specific dates (8, 15, 16, 17, 18) are highlighted in blue, indicating scheduled events. Below the calendar, there are two lines of text: "Zajęcia: Wyjątki. Generatory. Context manager. o 18:30" and "Zajęcia: Podział użytkowników na grupy według zainteresowań - wykład o 10:40".
- Ostatnio aktywne kursy (Recently active courses):** A sidebar listing recent courses with their descriptions and links:
 - Systemy rekomendacyjne: Algorytmy oparte na treści - wykład
 - Programowanie w języku Python: Podstawy języka Python. Typy modyfikowalne i niemodyfikowalne.
 - Systemy rekomendacyjne: Algorytmy oparte na treści - ćwiczenia
 - Systemy rekomendacyjne: Algorytmy wielorękich bandytów - wykład
 - Programowanie w języku Python: Łańcuchy, listy, krotki, zbiory i słowniki. Funkcje i lambdy.
 - Programowanie w języku Python: Obiektowość w Pythonie. Dziedziczenie.
- Nadchodzące aktywności (Upcoming activities):** A sidebar listing upcoming events with their descriptions and dates:
 - Algorytmy wielorękich bandytów - ćwiczenia (15.12.2022)
 - Podział użytkowników na grupy według zainteresowań - wykład (10.12.2022)
 - Wyjątki. Generatory. Context manager. (18.12.2022)
 - Obsługa plików. Dekoratory. Programowanie funkcyjne. (18.12.2022)

Rysunek 53: Strona główna aplikacji zawierająca plan zajęć

Wybierając zakładkę *Moje moduły* pokazana zostaje lista ze wszystkimi utworzonymi przez nauczyciela technicznego modułami, może on wyświetlić ich szczegóły, dokonać ich edycji, oraz w przypadku gdy nikt danego modułu jeszcze nie wykorzystał w aktywności edukacyjnej można go usunąć. Rysunek 54 przedstawia przykładową listę modułów.

Moje moduły		UTWÓRZ MODUŁ
Klasy w Pythonie #Informatyka	Liczba korzystających aktywności: 2	Data utworzenia: 23:30 29.05.2022 EDYTUJ SZCZEGÓŁY
Wideokonferencja #Dowolny	Liczba korzystających aktywności: 0	Data utworzenia: 17:47 01.12.2022 USUŃ EDYTUJ SZCZEGÓŁY
Podstawy bezpieczeństwa aplikacji webowych #Bezpieczeństwo	Liczba korzystających aktywności: 0	Data utworzenia: 12:14 03.12.2022 USUŃ EDYTUJ SZCZEGÓŁY
Quiz z ataków na strony internetowe #Informatyka	Liczba korzystających aktywności: 0	Data utworzenia: 15:22 05.12.2022 USUŃ EDYTUJ SZCZEGÓŁY

Rysunek 54: Lista modułów utworzonych przez nauczyciela technicznego

W celu utworzenia nowego modułu należy wybrać opcję *Utwórz moduł*. Wyświetlony zostanie formularz, którego pierwsza strona, przedstawiona na rysunku 55, wymaga wprowadzenia informacji o module tłumaczących jego zastosowanie oraz dostarczających instrukcji wykorzystania dla nauczyciela i uczniów. Należy też wybrać serwis, który ma zostać wykorzystany w module, dla każdego serwisu dostarczono instrukcję konfiguracji, wyświetlona po kliknięciu przycisku obok serwisu. Pola formularza są różne dla każdego serwisu, w przypadku Jupytera należy wprowadzić notebook, od którego uczniowie mają rozpoczęć pracę. W przykładzie wprowadzono notebook z podstaw kryptografii, którego zawartość zostanie pokazana w dalszej części.

Druga strona formularza, widoczna na rysunku 56, wymaga podania specyfikacji technicznej modułu, w szczególności trybu izolacji oraz wymagań maszyny wirtualnej. Wybrano tryb izolowany, w przypadku współdzielonego nauczyciel techniczny musiałby również wprowadzić dane dotyczące skalowania wymagań ze względu na liczbę użytkowników.

W końcowym etapie tworzenia modułu wyświetlane jest podsumowanie, przedstawione na rysunku 57, zawiera ono informacje pozwalające użytkownikowi na weryfikację wprowadzonych danych oraz tłumaczy co z utworzonym modułem można później zrobić.

Nazwa * — Szyfr Cezara i podstawy kryptoanalizy

Tematyka * — Informatyka

Opis — Wprowadzenie do szyfrowania, szyfr Cezara, szyfry monoalfabetyczne oraz podstawy kryptoanalizy.

Serwis * — Jupyter Notebook ▾ [POKAZ POMOC W KONFIGURACJI](#)

Konfiguracja serwisu

Startowy notebook * — kryptografia_lab1.ipynb [WYBIERZ PLIK](#)

Instrukcja dla nauczyciela

Normal Wymagana jest podstawowa wiedza z programowania w języku **Python**. Uczniowie powinni znać podstawowe pojęcia kryptografii.

Instrukcja dla studenta

Normal Przypomnij sobie podstawy z programowania w języku **Python** (pętle, zmienne, funkcje) oraz podstawowe pojęcia z kryptografią. Dołączony notebook będzie prowadził Cię krok po kroku, we wskazanych miejscach będziesz musiał samodzielnie zaimplementować pewne funkcjonalności.

Moduł publiczny

Rysunek 55: Formularz tworzenia modułu - informacje ogólne

Typ izolacji środowiska * — Izolowane

Minimalne wymagania sprzętowe maszyny wirtualnej

Liczba wirtualnych procesorów * — 1

Pamięć RAM [GB] * — 2

Pamięć nieulotna [GB] * — 8

Wymagana szerokość pasma [Mb/s] * — 256

Rysunek 56: Formularz tworzenia modułu - specyfikacja techniczna

Nowy moduł

Szyfr Cezara i podstawy kryptoanalizy

Wykorzystywany serwis: Jupyter Notebook

Moduł jest ustawiony jako publiczny.
Wszyscy nauczyciele będą mogli z niego skorzystać zaraz po utworzeniu.

Wybrano środowisko izolowane.
Tylko 1 użytkownik otrzyma dostęp do pojedynczej maszyny wirtualnej z uruchomionym modelem.

Pamiętaj

- Tworzenie modułu może chwilę potrwać
- Początkowo zaleca się ustawienie modułu jako prywatny i przetestowanie czy działa poprawnie korzystając z opcji 'Przetestuj moduł' w panelu modułu
- Moduł może zostać usunięty tylko gdy nikt z niego jeszcze nie skorzystał
- Żeby zablokować przyszłe korzystanie z modułu możesz wyłączyć opcję 'Moduł publiczny', jednak osoby które już go wybrali wciąż będą w stanie go użyć
- Zmiana typu izolacji i typu serwisu nie będzie możliwa, będziesz mógł jednak zmienić jego konfigurację, w szczególności zawartość plików i wymagania techniczne

[WRÓĆ](#) [UTWÓRZ MODUŁ](#)

Rysunek 57: Formularz tworzenia modułu - podsumowanie

Wszystkie publiczne moduły dostępne w systemie można zobaczyć w zakładce *Publiczne moduły*, w momencie gdy moduły potrzebne do przeprowadzenia aktywności zostały już utworzone można przejść do tworzenia kursu. W tym celu w panelu nawigacji nauczyciel musi wybrać opcję *Moje kursy*, która wyświetla stronę ze wszystkimi utworzonymi przez nauczyciela kursami, następnie kliknąć *Utwórz kurs*, jest ona bardzo podobna do strony z modułami dlatego nie zostanie tutaj pokazana.

Wyświetlony zostanie formularz, którego pierwsza strona, przedstawiona na rysunku 58, zawiera ogólne informacje o kursie takie jak początkową liczbę zajęć. Kolejne aktywności można dodać w dowolnej chwili po utworzeniu kursu. Druga strona formularza, widoczna na rysunku 59, wymaga skonfigurowania zajęć w liczbie wybranej we wcześniejszym kroku. Dla każdego z nich należy podać między innymi datę i czas trwania oraz wybrać potrzebne moduły. Podobnie jak w przypadku tworzenia modułu, na końcu wyświetlane jest podsumowanie z dodatkowymi informacjami, zaprezentowano je na rysunku 60.

W ramach przykładu pokazany zostanie przebieg aktywności, składającej się z utworzonego wcześniej modułu Jupytera, wideokonferencji oraz quizu. Wybrano opcję wymagania potwierdzenia wysłania linków do quizu, aby uczniowie nie mieli dostępu do pytań od początku zajęć.

Nazwa * Podstawy Kryptografii

Tematyka * Informatyka

Opis
Kurs ma na celu zapoznanie uczniów z najważniejszymi pojęciami z zakresu kryptografii.

Liczba zajęć * 2

Planowana liczba uczestników * 2

Hasło do dołączenia **** 

Kurs publiczny

Rysunek 58: Formularz tworzenia kursu - informacje ogólne

Zajęcia 1

Temat * Kryptografia w czasach starożytnych

Opis
W trakcie tych zajęć dowiesz się, czym jest szyfrowanie, zapoznasz się z podstawowymi metodami szyfrowania znanimi już w starożytności.

Data zajęć * 09/12/2022 

Czas rozpoczęcia * 11:20  Czas zakończenia * 12:50 

Moduły lekcyjne

Wideokonferencja  Wymagaj potwierdzenia przed wysłaniem uczniom linków

Quiz z podstaw kryptografii  Wymagaj potwierdzenia przed wysłaniem uczniom linków

Szyfr Cezara i podstawy kryptoanalizy  Wymagaj potwierdzenia przed wysłaniem uczniom linków

Instrukcje

Normal	B	<i>I</i>	<u>U</u>			
--------	----------	----------	----------	---	---	---

Otwórz dostarczony link, który przeniesie Cię do serwisu [Jupyter Notebooks](#). Zastaniesz w nim szablon do dzisiajszych zajęć. W trakcie zajęć będziemy wspólnie przechodzić przez wszystkie kroki, więc nie musisz się spieszyć z jego rozwijaniem. W razie jakichkolwiek problemów pytaj prowadzącego.

Zajęcia 2

Temat * Funkcje skrótu - wykład

Rysunek 59: Formularz tworzenia kursu - informacje o aktywnościach

Nowy kurs

Informacje ogólne Planowanie zajęć Podsumowanie

Podstawy Kryptografii

Nazwa aktywności	Wybrane moduły	Data aktywności
Kryptografia w czasach starożytnych	Wideokonferencja Quiz z podstaw kryptografii Szyfr Cezara i podstawy kryptoanalizy	09.12.2022 11:20 - 12:50
Funkcje skrótu - wykład	Wideokonferencja	13.12.2022 13:00 - 14:30

Pamiętaj

- Tworzenie kursu może trochę potrwać
- Po utworzeniu kursu będziesz mógł dodawać nowe aktywności oraz modyfikować jeszcze nieprzeprowadzone
- Po utworzeniu kursu w zakładce "Uczestnicy" będziesz mógł dodawać studentów

Rysunek 60: Formularz tworzenia kursu - podsumowanie

Po utworzeniu kursu nauczyciel zostaje przekierowany do strony z jego szczegółami, widocznej na rysunku 61. Poszczególne zakładki omówiono w sekcji 5.1.3, teraz skupimy się na aktywnościach zakładając, że uczniowie dołączyli już do kursu.

Podstawy Kryptografii

PRZEWIŃ DO NAJBLIŻSZEJ AKTYWNOSCI (11:20 09.12.2022)

AKTYWNOŚCI **UCZESTNICY** **EDYTOR KURSU**

Kryptografia w czasach starożytnych	09.12.2022 11:20 - 12:50
Funkcje skrótu - wykład	13.12.2022 13:00 - 14:30

Zajęcia odbędą się zgodnie z planem

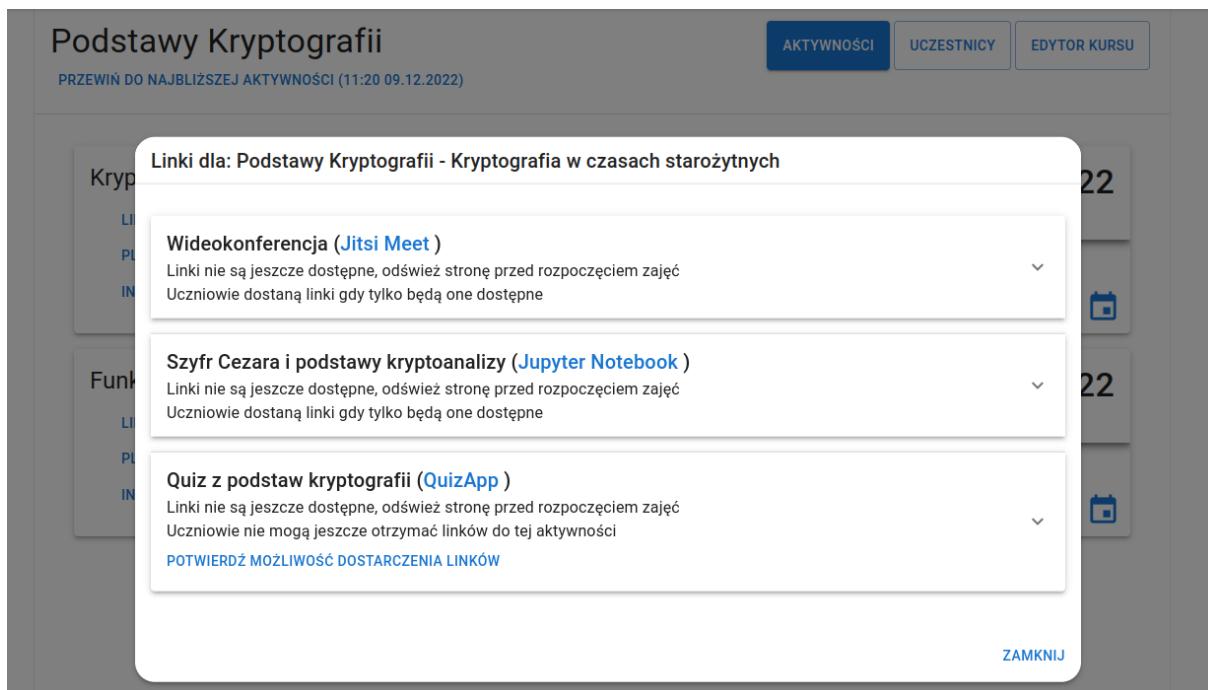
Rysunek 61: Strona ze szczegółami utworzonego kursu

W panelu aktywności widoczna jest data oraz czas trwania zajęć, w prawym dolnym rogu wyświetlany jest status zajęć, w tym przypadku status oznacza, że aktywność odbędzie się zgodnie z planem. Inne możliwe statusy to „aktywność odwołana“ - jeśli nauczyciel zrezygnował z jej prowadzenia, lub „aktywność już się odbyła“ - po skończeniu zajęć.

Każda z przyszłych aktywności posiada przycisk otwierający modal z linkami do wybranych przez nauczyciela modułów, widoczny na rysunku 62. Modal zawiera odnośniki do informacji o wykorzystywanych serwisach, w przypadku quizu dostępny jest również przycisk,

którego kliknięcie zezwoli systemowi na dostarczenie linków uczniom. Serwisy zostaną uruchomione przed rozpoczęciem zajęć i dopiero w momencie planowanego rozpoczęcia aktywności linki zostaną dostarczone uczestnikom kursu.

Przycisk *Instrukcje* prowadzi do strony z instrukcjami wprowadzonymi przez nauczyciela oraz nauczycieli technicznych, którzy są twórcami wykorzystywanych modułów. Przycisk *Pliki* otwiera stronę z plikami przypisanymi do aktywności, nauczyciel może przesłać w dowolnej chwili pliki, które mogą pobrać uczniowie, może to być przykładowo konspekt do zajęć. Po zakończeniu aktywności znajdą się tam również zapisane pliki z wynikami, zostanie to przedstawione w dalszej części.

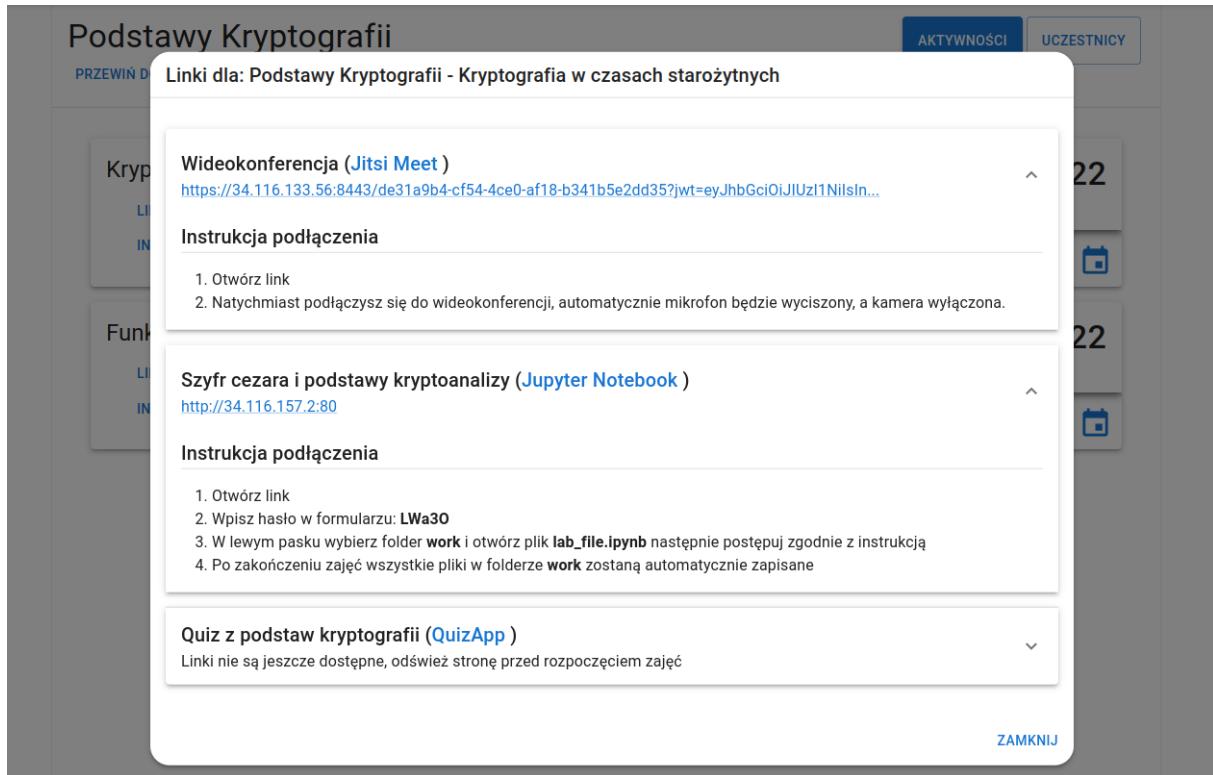


Rysunek 62: Modal z linkami przed czasem rozpoczęcia zajęć - punkt widzenia nauczyciela

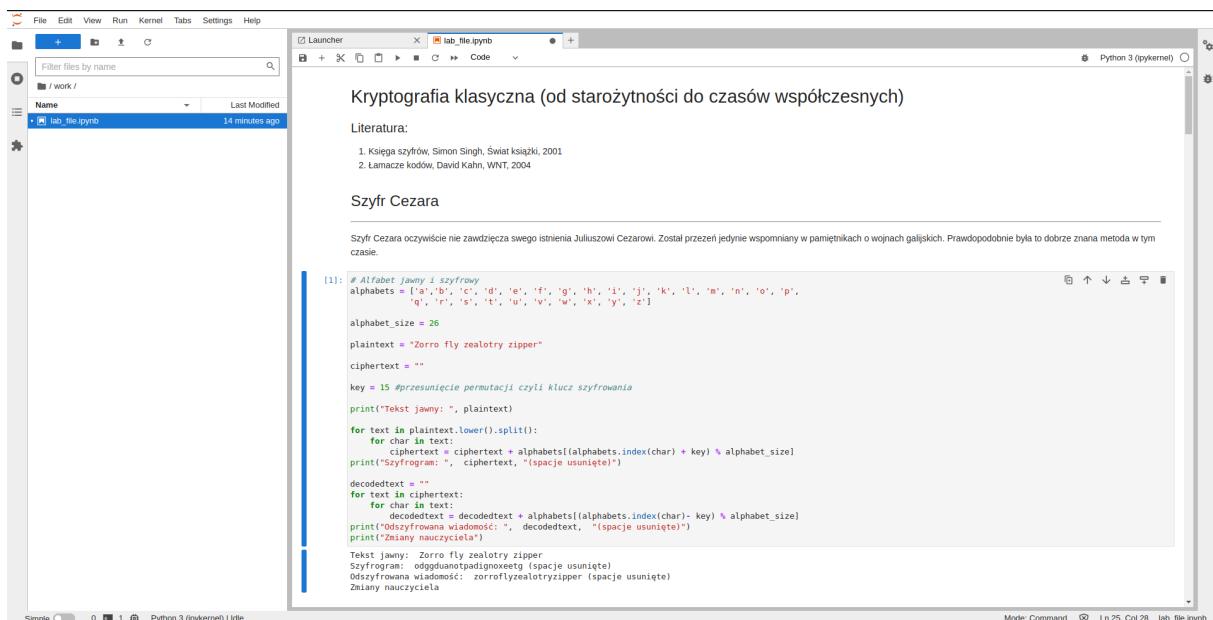
W momencie zaplanowanego rozpoczęcia aktywności modal zawiera linki do wszystkich modułów, dostarczone zostały również instrukcje podłączenia się. Widok z punktu widzenia ucznia został pokazany na rysunku 63, linki do quizu nie są dostępne, ponieważ nauczyciel nie potwierdził możliwości ich wysłania.

Nauczyciel oraz uczniowie muszą otworzyć linki w przeglądarce, a następnie rozpoznać pracę z modułami, zgodnie z podanymi instrukcjami. Poniżej przedstawiono wszystkie uruchomione w ramach pierwszej aktywności moduły:

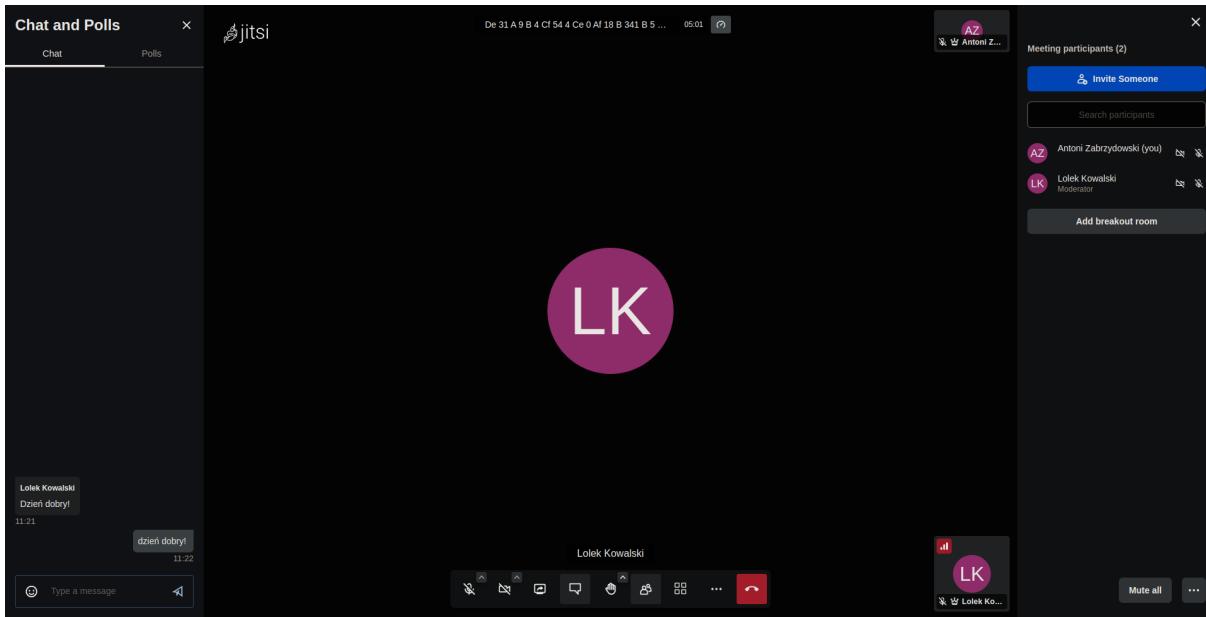
- skonfigurowany wcześniej moduł, działający na serwisie Jupyter (rysunek 64);
- wideokonferencję, udostępnioną przez serwis Jitsi Meet, zaprezentowaną z punktu widzenia ucznia (rysunek 65);
- quiz bazujący na serwisie QuizApp, do którego linki zostały dostarczone dopiero po udzieleniu zezwolenia przez nauczyciela (rysunek 66).



Rysunek 63: Modal z linkami w trakcie zajęć - punkt widzenia ucznia



Rysunek 64: Moduł bazujący na serwisie Jupyter, z otworzonym notebookiem do zajęć z kryptografi



Rysunek 65: Wideokonferencja bazująca na serwisie Jitsi Meet, z podłączonym nauczycielem oraz uczniem - punkt widzenia ucznia

Rysunek 66: Quiz bazujący na serwisie QuizApp

Po zakończeniu zaplanowanego czasu trwania aktywności dostęp do serwisów zostaje utracony, a wyniki zostają automatycznie zapisane. Uczniowie mają dostęp jedynie do własnych plików, a nauczyciel ma dostęp do wszystkich. Na rysunku 67 przedstawiono widok zapisanych plików z punktu widzenia nauczyciela, do którego konta przypisano jego własny uzupełniony notebook Jupytera oraz wyniki quizu wszystkich uczniów. W dolnej części widoku znajduje się lista z plikami z poszczególnych modułów, przypisanych do ich autorów.

Pliki dla: Podstawy Kryptografii - Kryptografia w czasach starożytnych

WRÓĆ DO KURSU

PUBLICZNE PLIKI MOJE PLIKI

Wyniki przypisane do konta nauczyciela

Nazwa pliku	Moduł	
workdirSnapshot.zip	Szyfr cezara i podstawy kryptoanalizy	Download
workdirSnapshot2.zip	Quiz z podstaw kryptografii	Download

Szyfr cezara i podstawy kryptoanalizy (Jupyter Notebook)

Imię i nazwisko studenta	Adres Email studenta	
Antoni Zabrzydowski	azabrzydowski@gmail.com	Download
Mela Nowak	mnowak@gmail.com	Download

workdirSnapshot2.zip Show all

Rysunek 67: Wyniki aktywności edukacyjnej w formie możliwych do pobrania plików - punkt widzenia nauczyciela

5.1.2. Interfejs panelu administratora

Po zalogowaniu się jako administrator wyświetlana jest lista użytkowników, przedstawiona na rysunku 68. Możliwe jest wyszukiwanie użytkowników posługując się imieniem i nazwiskiem, lub adresem e-mail. Z uwagi na liczbę uczniów w większych placówkach takich jak uniwersytety dostarczono również opcję filtracji według roku utworzenia konta oraz według roli.

Zarządzanie użytkownikami

NADCHODZĄCE AKTYWNOŚCI REPOZYTORIUM MASZYN WIRTUALNYCH ZARZĄDZANIE UŻYTKOWNIKAMI

LISTA UŻYTKOWNIKÓW KREATOR UŻYTKOWNIKÓW

Imię i nazwisko	Adres Email	Role	
Włodzimierz Biały	wbialy@gmail.com	Administrator	Szczegóły
Lolek Kowalski	lkowalski@gmail.com	Nauczyciel, Techniczny	Szczegóły
Bolek Zagórski	bzagorski@gmail.com	Nauczyciel	Szczegóły
Antoni Zabrzydowski	azabrzydowski@gmail.com	Student	Szczegóły
Mela Nowak	mnowak@gmail.com	Student	Szczegóły

Rysunek 68: Lista użytkowników z opcją wyszukiwania i filtracji

Klikając przycisk *Szczegóły* pokazany zostaje widok z szczegółowymi informacjami na temat danego użytkownika. W zależności od roli widoczne są tu różne opcje, dla każdego użytkownika dostępne są podstawowe informacje osobowe, które przedstawiono na rysunku 69. Dla nauczyciela oraz nauczyciela technicznego wyświetlane są również polityki, ograniczające możliwe do wykorzystania zasoby, przedstawiono je na rysunku 70. Administrator ma rów-

nież podgląd utworzonych przez nauczyciela kursów i modułów, pokazano to na rysunku 70. Z panelu szczegółów użytkownika możliwa jest edycja ról oraz polityk.

Szczegółowe dane użytkownika		WŁĄCZ TRYB EDYCJI	
Imię i nazwisko	Lolek Kowalski	Adres email	lkowalski@gmail.com
Data utworzenia konta	10.12.2022		
Wykorzystywane miejsce na dysku	62,5 kB		
Role użytkownika	<input checked="" type="radio"/> Nauczyciel <input type="radio"/> Techniczny		

Rysunek 69: Dane osobowe użytkownika

Polityki

Maksymalna liczba wirtualnych procesorów wykorzystywanych w 1 aktywności *	4
Maksymalna rozmiar pamięci RAM wykorzystywanej w 1 aktywności [GB] *	4
Maksymalna rozmiar pamięci nieulotnej wykorzystywanej w 1 aktywności [GB] *	32
Maksymalna szerokość pasma wykorzystywana w 1 aktywności [Mb/s] *	8192
Maksymalna liczba studentów w kursie *	50
Maksymalna długość pojedynczej aktywności [min] *	180
Maksymalna liczba jednocześnie uruchomionych środowisk testowych *	1
ZAAKTUALIZUJ POLITYKI	

Rysunek 70: Polityki przypisane do nauczyciela, wraz z możliwością edycji

Utworzone kursy			
Nazwa kursu	Tematyka	Data utworzenia	
Programowanie w języku Python	Informatyka	10.12.2022	
Systemy rekomendacyjne	Informatyka	10.12.2022	

Utworzone moduły			
Nazwa modułu	Wykorzystywany serwis	Data utworzenia	Liczba korzystających aktywności
Klasy w Pythonie	Jupyter Notebook	29.05.2022	2

Rysunek 71: Informacje o utworzonych przez użytkownika kursach i modułach

Po pierwszym uruchomieniu systemu dostępne jest wyłącznie konto administratora, który musi utworzyć konta użytkowników, pozwala na to widok przedstawiony na rysunku 72. Należy podać między innymi adres e-mail, na który po wybraniu przez użytkownika opcji zresetowania hasła zostanie wysłany token umożliwiający dokonanie tej akcji. Administrator nie wprowadza oraz nie ma dostępu do hasła użytkownika.

Zarządzanie użytkownikami	
LISTA UŻYTKOWNIKÓW	KREATOR UŻYTKOWNIKÓW
Utwórz pojedyncze konto użytkownika	
Imię * <input type="text" value="Rafał"/>	Nazwisko * <input type="text" value="Woźniak"/>
Adres email * <input type="text" value="rwozniak@gmail.com"/>	
Role <div style="display: flex; justify-content: space-around;"> <input type="button" value="Nauczyciel"/> <input type="button" value="Techniczny"/> </div>	
<input type="button" value="UTWÓRZ KONTO"/>	

Rysunek 72: Formularz tworzenia użytkowników

Zadaniem administratora po uruchomieniu systemu jest dodanie przynajmniej jednego typu maszyny wirtualnej, pozwala na to widok przedstawiony na rysunku 73. Przykładowa lista wszystkich utworzonych maszyn wirtualnych przedstawiona została na rysunku 74, administrator ma możliwość usunięcia maszyny w dowolnym momencie, może okazać się to przydatne gdyby okazało się, że koszty jej wykorzystania są zbyt wysokie.

Zarządzanie maszynami wirtualnymi

REPOZYTORIUM
KREATOR

Nazwa maszyny w systemie dostawcy usług chmurowych *	e2-standard-8
Liczba wirtualnych procesorów *	8
Liczba pamięci RAM [GB] *	32
Szerokość pasma [Mb/s] *	16384
Rozmiar dysku potrzebny do działania systemu operacyjnego [GB] *	10
Opis maszyny <div style="border: 1px solid #ccc; padding: 2px; display: flex; justify-content: space-around; align-items: center;"> Normal ⊕ B I U ⌚ ≡ ≡ ☒ </div> <p>Maszyna wirtualna ogólnego przeznaczenia z wysoką specyfikacją techniczną. Szczegóły: https://cloud.google.com/compute/docs/general-purpose-machines#e2_machine_types</p>	
ZAPISZ METADANE MASZYNY	

Rysunek 73: Formularz dodawania maszyn wirtualnych do repozytorium

Zarządzanie maszynami wirtualnymi

REPOZYTORIUM
KREATOR

Nazwa	Liczba vCpu	Pamięć RAM [GB]	Szerokość pasma [Mb/s]	
e2-medium	2	4	2048	📄 刪
e2-standard-4	4	16	8192	📄 刪
e2-standard-8	8	32	16384	📄 刪

Rysunek 74: Lista maszyn wirtualnych możliwych do wykorzystania przez system

Ostatnią funkcjonalnością oferowaną administratorowi przez system jest możliwość monitorowania nadchodzących aktywności edukacyjnych oraz ich ewentualnego odwoływania, przedstawiono to na rysunku 75.

Nadchodzące aktywności

REPOZYTORIUM
KREATOR

Nazwa	Nauczyciel	Data zajęć	
Podział użytkowników na grupy według zainteresowań - wykład	Lolek Kowalski (lkowalski@gmail.com)	10:40 - 12:10 12.12.2022	ODWOŁAJ AKTYWNOŚĆ
Wyjątki. Generatory. Context manager.	Lolek Kowalski (lkowalski@gmail.com)	18:30 - 20:30 12.12.2022	ODWOŁAJ AKTYWNOŚĆ

Rysunek 75: Lista nadchodzących aktywności edukacyjnych z możliwością ich odwołania

Jak już wspomniano, część wymagań funkcjonalnych związanych z monitorowaniem systemu została spełniona przez wybór platformy Google Cloud jako dostawcy usług chmurowych, pozwala ona w szczególności na sprawdzenie kosztów, stanu plików, a także stanu maszyn wirtualnych, wraz z historią ich uruchamiania.

5.1.3. Pozostałe widoki

W sekcji zebrano i pogrupowano, zgodnie z podziałem na podsystemy funkcjonalne opisany w sekcji 2.3, bardziej istotne z punktu widzenia użytkownika widoki.

a) Podsystem zarządzania modułami

Nauczyciel techniczny chcąc utworzyć moduł musi wybrać serwis. Lista dostępnych serwisów wraz z ogólnymi opisami oraz linkami do szczegółowych źródeł jest dostępna w zakładce *Serwisy*, przedstawionej na rysunku 76.

Dostępne serwisy

Docker
Serwis umożliwia uruchamianie kontenerów Dockera na podstawie publicznych obrazów dostarczających dowolną funkcjonalność. Więcej informacji można znaleźć na [oficjalnej stronie](#).
Główne zastosowania:

- Uruchamianie dowolnego oprogramowania, dostarczającego funkcjonalność za pośrednictwem przeglądarki.

W przypadku trybu izolowanego wyniki mogą być dostępne dla wszystkich uczestników zajęć, w przypadku współdzielonego tylko dla nauczyciela. Instrukcja wykorzystania powinna zawierać informacje o sposobie użycia wyników, zależną od wykorystanego oprogramowania.

Jitsi Meet
Serwis pozwala na przeprowadzenie wideokonferencji. Więcej informacji można znaleźć na [oficjalnej stronie](#).
Główne zastosowania:

- Wideokonferencja z możliwym użyciem mikrofonu, kamerki, czatu i udostępnianiem ekranu.

Jupyter Notebook
Serwis pozwala na operowanie na notebookach Jupytera. Więcej informacji można znaleźć na [oficjalnej stronie](#).
Główne zastosowania:

- Zajęcia z programowania, wykorzystujące język Python;

Rysunek 76: Informacje o zintegrowanych serwisach wraz z głównymi zastosowaniami

W celu prezentacji funkcjonalności środowisk testowych zostanie pokazany test serwisu pominiętego w sekcji 5.1.1 - Dockera. Obraz Dockera zaprezentowany w przykładzie był używany na zajęciach z Inżynierii Bezpieczeństwa prowadzonych na naszym wydziale, każdy ze studentów musiał mieć zainstalowane oprogramowanie Docker na swoim komputerze i musiał sam potrafić go uruchomić. Z wykorzystaniem naszego produktu wszystko to mogłoby zostać zrobione automatycznie, a uczniowie musieliby jedynie wejść w dostarczony link, nie tracąc przy tym czasu zajęć. Obraz udostępnia stronę internetową, pozwalającą na zapoznanie się z atakami takimi jak SQL Injection. Rysunek 77 przedstawia konieczne do podania przez nauczyciela technicznego parametry.

Serwis * Docker POKAŻ POMOC W KONFIGURACJI

Konfiguracja serwisu

Nazwa publicznego obrazu * bkimminich/juice-shop

Oczekiwany czas uruchomienia serwisu [s] * 60

Szacowany rozmiar obrazu [MB] * 177

Port do serwisu (wewnętrz kontenera) * 80

Ścieżka do wyników (wewnętrz kontenera)

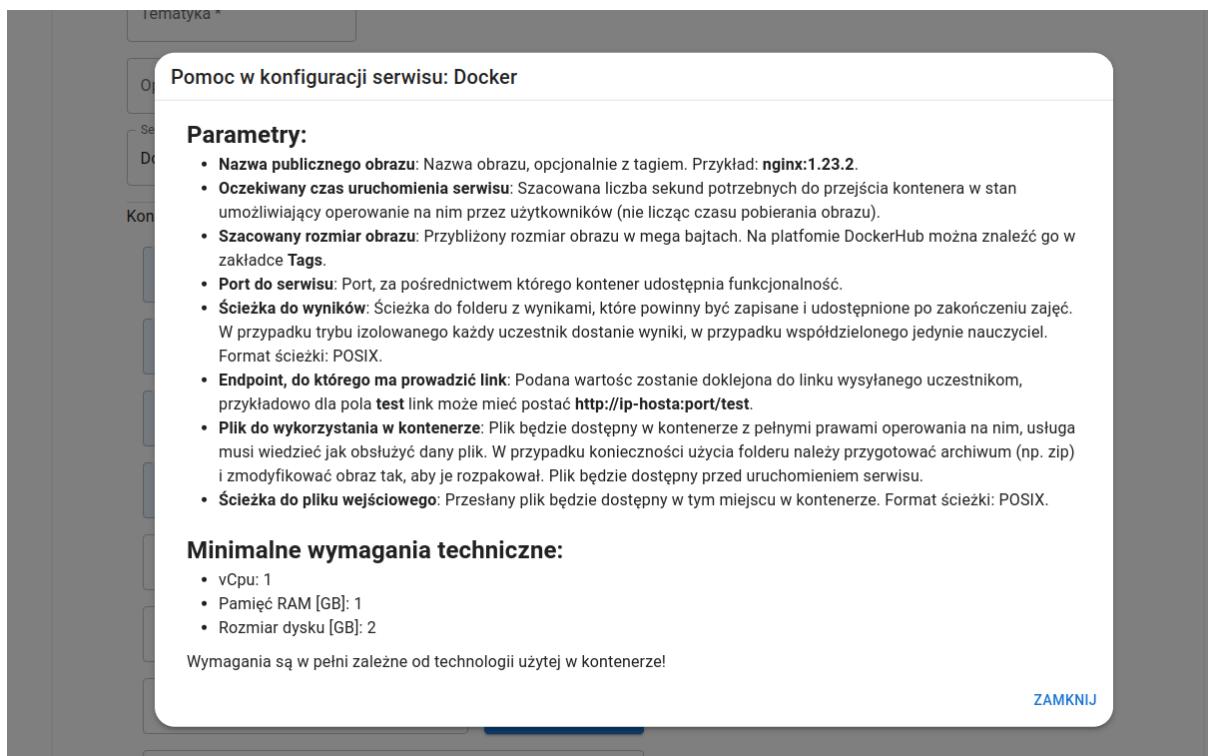
Endpoint, do którego ma prowadzić link

Plik do wykorzystania w kontenerze

Ścieżka do pliku wejściowego (wewnętrz kontenera)

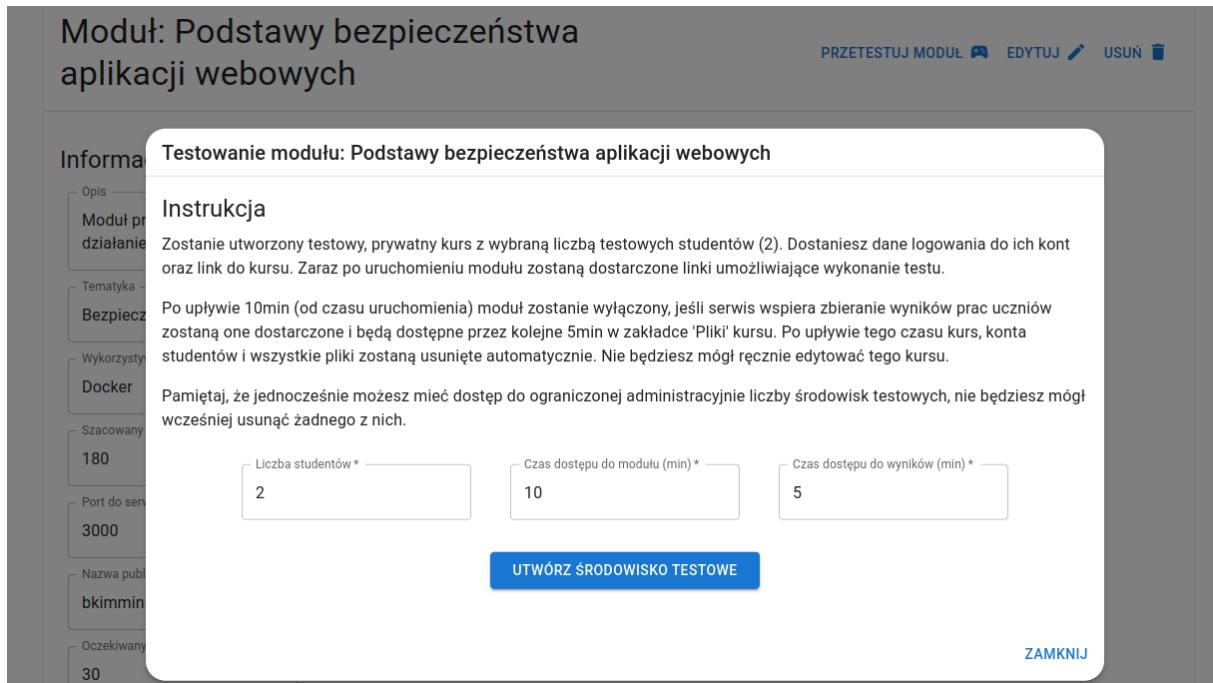
Rysunek 77: Formularz tworzenia modułu - konfiguracja serwisu Docker

Konfiguracja serwisu podczas tworzenia modułu może być skomplikowanym zadaniem, dlatego w panelu tworzenia modułu można wyświetlić modal z pomocą. Modal dla serwisu Docker przedstawiono na rysunku 78, zawiera on opis parametrów w dynamicznym formularzu, oraz minimalne wymagania techniczne serwisu, które w przypadku Dockera zależą od samego obrazu.



Rysunek 78: Instrukcja konfiguracji serwisu Docker

Jak już wspomniano w sekcji 3.6.4 po utworzeniu modułu zalecane jest wykonanie testu, w celu sprawdzenia czy został on poprawnie skonfigurowany. Opcja przetestowania modułu jest dostępna zarówno dla nauczycieli technicznych, jak i zwykłych nauczycieli chcących sprawdzić czy moduł spełnia ich oczekiwania pod kątem dydaktycznym. Na stronie z szczegółowymi informacjami o module widoczny jest przycisk *Przetestuj moduł*, który otwiera modal opisujący zasadę działania środowiska testowego, przedstawiono go na rysunku 79.



Rysunek 79: Modal z formularzem tworzenia środowiska testowego, wraz z instrukcją jego działania

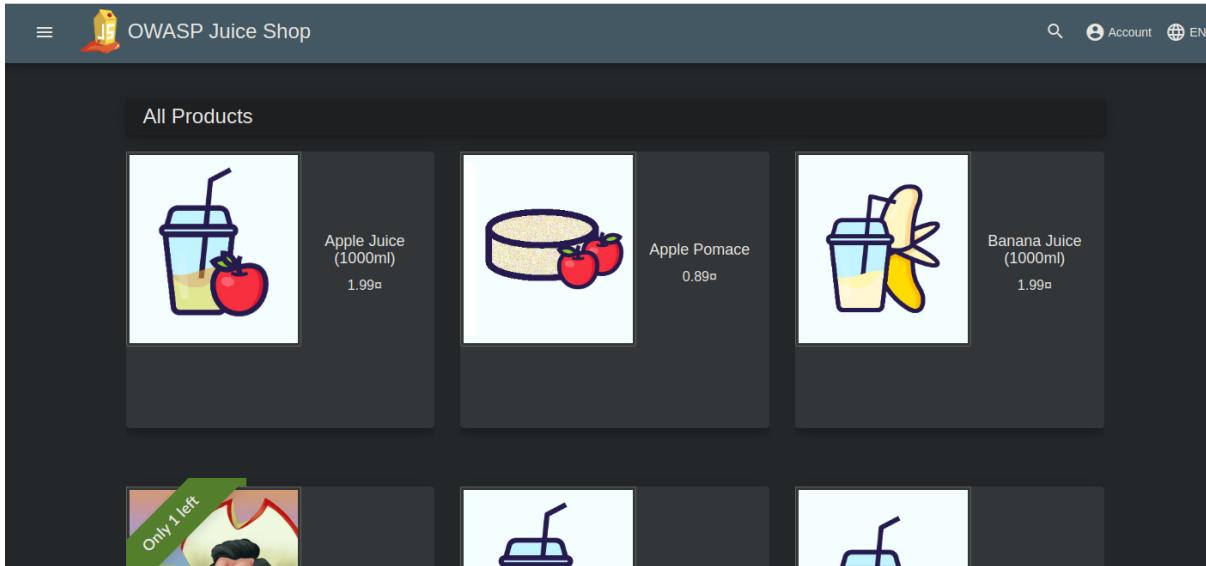


Rysunek 80: Fragment modalu tworzenia środowiska testowego, zawierający dane logowania do utworzonych kont uczniów

Po wprowadzeniu danych w formularzu zostaje utworzony testowy kurs, a w modalu znajdują się wyświetlane dane logowania do kont studentów, które po zakończeniu testu zostaną

usunięte automatycznie. Fragment modalu po utworzeniu środowiska testowego przedstawiono na rysunku 80.

W efekcie utworzony zostaje kurs z pojedynczą aktywnością, wyglądający analogicznie do kursu przedstawionego na rysunku 61. Po uruchomieniu modułu użytkownik na własne konto oraz na utworzone konta studentów dostaje link, dzięki któremu może go przetestować przez określony czas. Rysunek 81 przedstawia stronę internetową dostępną za pośrednictwem modułu bazującego na serwisie Docker.



Rysunek 81: Moduł do nauki bezpieczeństwa stron internetowych, bazujący na serwisie Docker

b) Podsystem zarządzania kursami i zajęciami

Po utworzeniu kursu nie ma w nim żadnych studentów, nauczyciel może ich dodać ręcznie wykorzystując adresy e-mail, lub może wysłać im link do dołączenia do kursu. Rysunek 82 przedstawia widok z uczestnikami kursu.

Imię i nazwisko	Adres email	Data dołączenia
Antoni Zabrzydowski	azabrzydowski@gmail.com	09.12.2022
Mela Nowak	mnowak@gmail.com	09.12.2022

Rysunek 82: Lista uczestników kursu

Student może dołączyć do kursu wykorzystując udostępniony przez nauczyciela link oraz hasło. Rysunek 83 przedstawia widok wyświetlony po otworzeniu linku przez ucznia.



Rysunek 83: Widok umożliwiający dołączenie do kursu

Nauczyciel może również edytować kurs, w szczególności zmienić jego nazwę czy opis. Może on także dodawać nowe aktywności do utworzonego wcześniej kursu, formularze są podobne do pokazanych na rysunkach 58 oraz 59. Ponadto nauczyciel ma możliwość odwołania zajęć lub ich usunięcia, w pierwszym przypadku aktywność będzie widoczna na stronie kursu, z informacją, że została odwołana, w drugim przypadku zostanie ona całkowicie usunięta. Rysunek 84 przedstawia fragment edytora kursu, z otwartą zakładką usuwania aktywności.

Rysunek 84: Edytor kursu - zakładka umożliwiająca odwoływanie zajęć

Uczeń może wyświetlić wszystkie instrukcje dołączone do aktywności edukacyjnej, zarówno ogólną instrukcję wprowadzoną przez nauczyciela, jak i poszczególne instrukcje do modułów dostarczone przez nauczyciela technicznego. Rysunek 85 przedstawia widok z przykładowymi instrukcjami do modułów z punktu widzenia studenta. Nauczycielowi wyświetlane są dodatkowo dedykowane instrukcje, wprowadzone przez twórcę modułu.

Instrukcje dla: Podstawy Kryptografii - Kryptografia w czasach starożytnych

[WRÓĆ DO KURSU](#)
[INSTRUKCJE OD NAUCZYCIELA](#)
[INFORMACJE O UŻYWANYCH MODUŁACH](#)

Szyfr cezara i podstawy kryptoanalizy ([Jupyter Notebook](#))

Instrukcje dla studenta

Przypomnij sobie podstawy z programowania w języku Python (pętle, zmienne, funkcje) oraz podstawowe pojęcia z kryptografią. Dołączony notebook będzie prowadził Cię krok po kroku, we wskazanych miejscach będziesz musiał samodzielnie zaimplementować pewne funkcjonalności.

Quiz z podstaw kryptografii ([QuizApp](#))

Instrukcje dla studenta

Quiz składa się z 3 pytań jednokrotnego wyboru, na odpowiedź będziesz miał **90 sekund**. Warto wcześniej zapoznać się z wykładem.

Rysunek 85: Instrukcje do zajęć - widok ucznia, zakładka z instrukcjami wprowadzonymi przez nauczyciela technicznego

Uczeń ma możliwość dodania najważniejszych dla niego plików z różnych zajęć do listy ulubionych, żeby mieć do nich łatwy dostęp w jednym miejscu. Rysunek 86 przedstawia widok z listą ulubionych plików ucznia z trzech różnych kursów.

Moje pliki

Nazwa pliku	Nazwa kursu	Data utworzenia
-------------	-------------	-----------------

program-demonstrujacy.py	Programowanie w języku Python	10.12.2022	
--------------------------	-------------------------------	------------	--

konspekt.pdf	Systemy rekomendacyjne	10.12.2022	
--------------	------------------------	------------	--

workdirSnapshot.zip	Podstawy Kryptografii	09.12.2022	
---------------------	-----------------------	------------	--

Rysunek 86: Lista ulubionych plików ucznia

5.2. Możliwości rozwoju

Możemy wyróżnić dwie główne ścieżki potencjalnego rozwoju systemu: dodawanie nowych funkcjonalności oraz poprawki implementacji aktualnego rozwiązania. W sekcji zostaną dokładniej opisane opcje wchodzące w skład każdej z nich.

5.2.1. Nowe funkcjonalności

a) Powiadomienia w systemie.

Powiadomienia dotyczące odwołania zajęć były jednym z wymagań funkcjonalnych systemu, z uwagi na brak czasu nie udało się tego zrealizować. Aktualnie nauczyciel powinien poinformować uczniów np. drogą mailową o odwołaniu zajęć, na stronie kursu jest widoczna jedynie ikonka informująca, że się one nie odbędą.

b) Podział studentów na grupy.

Aktualnie możliwe są dwa tryby pracy modułów, izolowany, w którym każdy student dostaje link do dedykowanej maszyny wirtualnej oraz współdzielony, w którym wszyscy uczestnicy kursu dostają ten sam link. Można by było zrobić coś pośredniego, czyli pozwolić nauczycielowi na określenie jacy uczniowie mają dostać ten sam link w celu współpracy na zajęciach. Przy aktualnie zaimplementowanych serwisach mogłoby to mieć zastosowanie przy wideokonferencji (uczniowe mogliby rozmawiać tylko ze sobą, bez przeszkadzania innym) oraz przy serwisie Docker, w zależności od użytego obrazu.

c) Opcja uruchomienia serwisu bazując na zapisanych wcześniej wynikach.

Mogłoby to mieć zastosowanie między innymi przy serwisie Jupyter. Uczeń mógłby chcieć powtórzyć zagadnienia przedstawione na już zakończonych zajęciach, wykorzystując swój wcześniejszy uzupełniony notebook.

d) Opcja przedłużenia czasu trwania zajęć.

Wszystkie serwisy są automatycznie zatrzymywane w momencie planowanego początkowo czasu zakończenia zajęć, uniemożliwia to przykładowo dokończenie pracy na przerwie między zajęciami. Można by było pozwolić nauczycielowi na określenie chwilę przed końcem aktywności, czy powinna być ona przedłużona i jeśli tak to na jak długo.

e) Opcja wykorzystania maszyn wirtualnych wyposażonych w GPU.

Mogłoby to się okazać ważne przykładowo dla modułów z uczenia maszynowego bazujących na serwisie Jupyter. Aktualnie administrator nie ma możliwości definiowania specyfikacji technicznej dotyczącej GPU.

f) Integracja kolejnych serwisów.

Zakres funkcjonalności systemu będzie rósł z każdym kolejnym zintegrowanym serwisem, przykładowo przydatne mogłyby się okazać ankiety lub interaktywna tablica dla nauczycieli.

5.2.2. Poprawki implementacji aktualnego rozwiązania

a) Widok mobilny i zwiększoną responsywność *Aplikacji Webowej*.

Na ten moment zaimplementowane jest wsparcie jedynie dla widoku przeznaczonego dla dużych ekranów (rozdzielcość HD bądź większa).

b) Optymalizacja Serwera.

Z uwagi na brak czasu niektóre funkcjonalności zostały zaimplementowane w nieoptymalny sposób. Przykładowo zapytanie o listę dostępnych kursów zawsze zwraca wszystkie kursy, można by było dodać tutaj paginację. Dodatkowo należałoby przeprowadzić testy wydajności związane z obsługą bazy danych i w razie potrzeby założyć indeksy na odpowiednie atrybuty.

c) Dokładniejsze sumowanie czasu uruchamiania wielu serwisów na jednej maszynie wirtualnej.

Aktualnie brana jest pod uwagę zwykła suma oczekiwanej czasu uruchamiania poszczególnych serwisów, niektóre z nich bazują jednak na tym samym oprogramowaniu. W efekcie kolejny serwis na tej samej maszynie zostanie uruchomiony szybciej niż w przypadku gdy byłby jedynym serwisem. Dokładniejsze liczenie sumy pozwoliłoby na skrócenie czasu działania maszyny wirtualnej, a co za tym idzie na redukcję kosztów.

d) Zastosowanie rozproszonej kolejki zadań w *Orkiestratorze*.

Zwiększyłoby to wydajność i niezawodność *Orkiestratora*, a także ułatwiłoby jego replikację.

5.3. Ocena rezultatów pracy oraz wnioski

W wyniku prac zespołu zrealizowana została większość wymagań produktowych. Powstały system dostarcza typowe funkcje LMS takie jak organizację nauczania czy plany zajęć i jest możliwy do wdrożenia w placówkach o ograniczonych zasobach sprzętowych. Z systemem zintegrowano serwisy umożliwiające prowadzenie szerokiego zakresu różnorodnych zajęć online, nie tylko w placówkach informatycznych.

Utworzony rdzeń systemu umożliwia nieskomplikowaną integrację kolejnych serwisów i zwiększanie funkcjonalności produktu, bez konieczności wprowadzania w nim modyfikacji. Dzięki architekturze mikroserwisowej, w której każdy z zintegrowanych serwisów uruchamiany jest na innym sprzęcie niż *Serwer* i *Orkiestrator*, możliwe jest horyzontalne skalowanie systemu do właściwie dowolnych rozmiarów. Dzięki temu otrzymano również system, który jest w bardzo dużym stopniu odporny na awarie, ponieważ ewentualne błędy występujące w poszczególnych zintegrowanych serwisach nie propagują się na pozostałe komponenty.

Wzorzec MDA okazał się bardzo przydatny w implementacji procesu wyboru optymalnej maszyny wirtualnej, bez komplikowania interfejsu nauczyciela technicznymi detalami. Dzięki obecnym w nim translacjom otrzymano uporządkowaną i łatwą w rozbudowie implementację.

Głównym zagrożeniem związanym z wykorzystaniem produktu są koszty naliczane przez platformę Google Cloud. Wskazano wytyczne prowadzące do ich redukcji, lecz wdrożenie ich nie gwarantuje utrzymania kosztów na szacowanym poziomie. Z tego powodu zaleca się ciągłe monitorowanie naliczanych opłat i podejmowanie odpowiednich działań w celu ochrony budżetu. Integracja z Google Cloud została zaimplementowana w sposób umożliwiający relatywnie prostą zmianę dostawcy, jednak przykładowo użycie chmury akademickiej zamiast publicznej mogłoby wiązać się z istotnym nakładem pracy.

Materiały źródłowe

- [1] R. L. Gandia, S. Zieliński, and M. Konieczny. Model-based approach to automated provisioning of collaborative educational services. In M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, editors, *Computational Science – ICCS 2021*, pages 640–653, Cham, 2021. Springer International Publishing.
- [2] R. Llopis Gandia. On-demand provisioning of educational cloud-based services, 07 2020.
- [3] B. Meijer, L. Hochstein, and R. Moser. *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. O'Reilly, 2017.
- [4] Dokumentacja Docker Compose. <https://docs.docker.com/compose/compose-file/>.
- [5] Dokumentacja i instrukcja instalacji serwisu Jitsi Meet. <https://jitsi.github.io/handbook/docs/devops-guide/devops-guide-docker/>.
- [6] Dokumentacja i instrukcja instalacji serwisu Jupyter. <https://t1jh.jupyter.org/en/latest/install/custom-server.html>.
- [7] Dokumentacja i kod źródłowy serwisu QuizApp. <https://github.com/SafdarJamal/quiz-app>.
- [8] Dokumentacja maszyn wirtualnych dostępnych na platformie Google Cloud. <https://cloud.google.com/compute/docs/machine-resource>.
- [9] Dokumentacja modułu ansible.builtin.file. https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html.
- [10] Instrukcja podłączania do maszyn wirtualnych w Compute Engine. <https://cloud.google.com/compute/docs/instances/connecting-advanced#linux-and-macos>.
- [11] Instrukcja wykorzystania podpisanych cyfrowo adresów URL do bezpośredniego transferu plików między klientem a kubelkiem w systemie Google Cloud Storage. <https://cloud.google.com/blog/products/storage-data-transfer/uploading-images-directly-to-cloud-storage-by-using-signed-url>.
- [12] Kod źródłowy projektu. <https://github.com/mikkolaj/swozo>.
- [13] Kod źródłowy projektu Sozisel. <https://github.com/Bombardierzy/sozisel>.
- [14] Kod źródłowy prototypu. <https://github.com/llopisga/cloud-orchestration>.
- [15] Konfiguracja reverse proxy (Nginx) umożliwiająca użycie protokołu HTTPS. <https://docs.nginx.com/nginx/admin-guide/security-controls/terminating-ssl-http>.
- [16] Kontrola dostępu do poszczególnych endpointów w środowisku Spring. <https://www.baeldung.com/spring-security-method-security>.

- [17] Mechanizm automatycznego skalowania maszyn wirtualnych w infrastrukturze Google Cloud. <https://cloud.google.com/compute/docs/autoscaler>.
- [18] Mechanizm autoryzacji na podstawie ról w środowisku Spring. <https://www.baeldung.com/role-and-privilege-for-spring-security-registration>.
- [19] Metody komunikacji w środowiskach mikroserwisowych. <https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture>.
- [20] Metody pobierania dynamicznych danych w aplikacjach webowych. <https://nextjs.org/docs/basic-features/data-fetching/overview>.
- [21] Modelowanie relacji między encjami z wykorzystaniem platformy Hibernate. <https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate>.
- [22] Obsługa asynchronicznych operacji w Java. <https://www.baeldung.com/java-completablefuture>.
- [23] Opis Apache Tomcat, porównanie z alternatywnymi rozwiązaniami. <https://www.jrebel.com/blog/what-is-apache-tomcat>.
- [24] Opis serwisu Docker z wyjaśnieniem jak działają obrazy Dockera. <https://docker-curriculum.com>.
- [25] Pamięć podręczna w bibliotece React Query. <https://ilxanlar.medium.com/fetch-cache-and-update-data-effortlessly-with-react-query-445e799a84e4>.
- [26] Persystencja w Hibernate i działanie EntityManagera. <https://www.baeldung.com/jpa-hibernate-persistence-context>.
- [27] Porównanie algorytmów wyliczania funkcji skrótu z hasel użytkowników. <https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaaf41598e>.
- [28] Porównanie standardu GraphQL z REST. <https://www.apollographql.com/blog/graphql/basics/graphql-vs-rest>.
- [29] Porównanie uwierzytelniania na podstawie JWT i ciasteczek. <https://developer.okta.com/blog/2022/02/08/cookies-vs-tokens>.
- [30] Porównanie środowisk programistycznych Angular i React. <https://www.trio.dev/blog/angular-vs-react>.
- [31] Strefy dostępu i regiony w środowiskach chmurowych. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>.
- [32] Szyfrowanie danych w Google Cloud Storage. <https://cloud.google.com/docs/security/encryption/default-encryption>.

-
- [33] Szyfrowanie komunikacji w infrastrukturze Google Cloud. <https://cloud.google.com/docs/security/encryption-in-transit>.
 - [34] Tworzenie obrazów Dockerowych dla aplikacji Spring Boot. <https://spring.io/guides/gs/spring-boot-docker/>.
 - [35] Wprowadzenie do standardu Material Design. <https://m2.material.io/design/introduction>.
 - [36] Wykorzystanie JWT do uwierzytelniania. <https://jwt.io/introduction>.
 - [37] Wykorzystanie standardu OpenApi w środowisku Spring. <https://www.baeldung.com/spring-rest-openapi-documentation>.
 - [38] Zarządzanie kluczami SSH w Google Cloud. <https://cloud.google.com/compute/docs/connect/add-ssh-keys#metadata>.
 - [39] Zasady działania silników wyszukiwania i web crawler'ów. <https://nextjs.org/learn/seo/introduction-to-seo/search-systems>.
 - [40] Zasady projektowania skalowalnych i odpornych na błędy systemów. <https://www.reactivemanifesto.org/>.
 - [41] Zastosowanie oprogramowania RabbitMQ w środowiskach mikroserwisowych. <https://www.cloudamqp.com/blog/why-use-rabbitmq-in-a-microservice-architecture.html>.
 - [42] Środowiska testowe (sandboxy) w systemach informatycznych. <https://www.proofpoint.com/us/threat-reference/sandbox>.
 - [43] Object Management Group. Oficjalny poradnik Model Driven Architecture (MDA). <https://www.omg.org/cgi-bin/doc?ormsc/14-06-01>.
 - [44] I. Sacevski and J. Veseli. Introduction to model driven architecture (mda). 2007.
 - [45] H. Schildt. *Java: The Complete Reference, Eleventh Edition*. O'Reilly, 2019.