



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PROJEKT DYPLOMOWY

**AUTOMATYZACJA TWORZENIA WIRTUALNEGO
LABORATORIUM SIECI PROGRAMOWALNYCH**

AUTOMATED CREATION OF VIRTUAL LAB ON PROGRAMMABLE NETWORKS

Autorzy:
Kierunek studiów:
Typ studiów:
Opiekun pracy:

Arkadiusz Kraus, Mateusz Naróg
Informatyka
Stacjonarne
dr inż. Sławomir Zieliński

Kraków, 2021

Uprzedzony 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 2006 r. Nr 90, poz. 631 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, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....

PODPIS

Spis treści

1	Cel prac i wizja produktu	4
1.1	Wstęp	4
1.2	Wizja	4
1.2.1	Laboratorium	4
1.2.2	Koncepcja systemu	5
1.2.3	Sieci programowalne	6
1.3	Istniejące rozwiązania	6
1.4	Analiza zagrożeń	7
2	Zakres funkcjonalności	8
2.1	Kontekst użytkowania	8
2.2	Wymagania funkcjonalne	9
2.3	Wymagania нефункционалне	9
3	Wybrane aspekty realizacji	11
3.1	Koncept technologiczny	11
3.2	Architektura systemu	12
3.3	Komponenty systemu	13
3.3.1	Szablony laboratoriów	13
3.3.2	Worker	14
3.3.3	Aplikacja zarządzająca	16
3.3.4	Load Balancer	18
3.3.5	Uwierzytelnianie i autoryzacja	18
3.4	CI/CD	19
3.5	Środowiska programistyczne	19
3.6	Chmura obliczeniowa	20
3.7	Szablon laboratorium P4	20
3.8	Testy aplikacji	21
3.8.1	Backend oraz Worker	22
3.8.2	Frontend	22
4	Organizacja pracy	23
4.1	Planowanie i przebieg pracy	23
4.2	Techniczna organizacja pracy	23
4.3	Osoby zaangażowane w projekcie	24
4.4	Podział zadań	24
4.5	Zrealizowane etapy pracy	25
5	Wyniki projektu	27
5.1	Produkt końcowy	27
5.2	Realizacja wymagań funkcjonalnych	30
5.3	Realizacja wymagań нефункционалных	31
5.4	Analiza zagrożeń	33
5.5	Rozwój systemu	34
5.6	Podsumowanie	36
A	Przykładowy szablon laboratorium	37

1. Cel prac i wizja produktu

1.1. Wstęp

Przygotowanie środowiska pracy nieraz wiąże się z instalacją oraz konfiguracją niezliczonej ilości serwisów, modułów i narzędzi. Z tego względu proces automatyzacji jest coraz bardziej pożądany, a wręcz nieunikniony, ponieważ na co dzień mamy styczność z coraz bardziej złożonymi systemami komputerowymi. Manualne wdrożenie oraz zarządzanie może okazać się kłopotliwe w wielu przypadkach.

Nie bez powodu coraz większą popularność zdobywa więc podejście “as a Service”, w którym to infrastruktura, aplikacje oraz inne narzędzia udostępniane są użytkownikowi na żądanie, w najczęstszym przypadku przez dostawców chmur. Podejście to może okazać się kluczowe w czasach, gdy system edukacji oraz inne dziedziny muszą być realizowane w formie zdalnej i gdzie nie każdy uczeń/pracownik ma dostęp do oprogramowania oraz wymaganych zasobów na własnym sprzęcie.

Poprzez elastyczność, prostotę zarządzania oraz dostarczane możliwości, kolejnym tematem, który cieszy się coraz większym zainteresowaniem są sieci programowalne. W dzisiejszych czasach bardzo ważne jest, aby od początku procesu edukacyjnego mieć też styczność z najnowszymi technologiami oraz trendami. Aktualnie jednak narzędzia do sieci programowalnych są we wczesnej fazie rozwoju i przygotowanie środowiska na pojedyncze zajęcia laboratoryjne może być problematyczne.

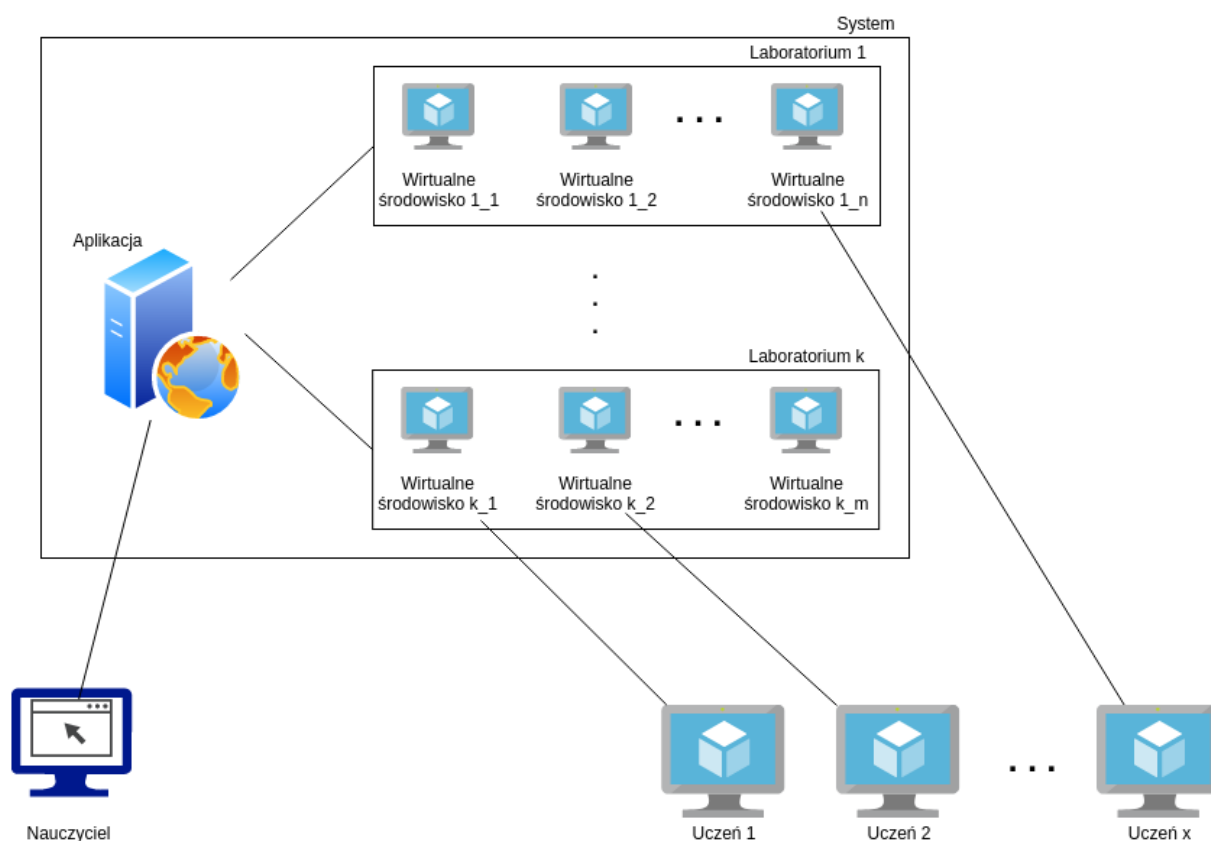
Dlatego też, w niniejszej pracy podjęto się opracowania produktu, którego celem będzie automatyzacja procesu tworzenia wirtualnego laboratorium sieci programowalnych, a więc stworzenie “Laboratorium as a Service”. Głównym założeniem będzie ułatwienie procesu edukacyjnego, zarówno nauczycielowi prowadzącemu zajęcia, jak również uczniom.

1.2. Wizja

Niniejszy rozdział przedstawia wysokopoziomowy zarys wizji gotowego produktu, wyjaśniając również podstawowe pojęcia używane w systemie. Opisuje on jak aplikacja będzie wyglądała dla użytkownika zewnętrznego.

1.2.1. Laboratorium

Laboratorium definiujemy jako gotowe środowisko z zainstalowanym oprogramowaniem, które będzie niezbędne do wykonania określonego zadania przez użytkownika, wraz z opisem tego zadania. Dostęp do laboratorium będzie się odbywać za pomocą protokołów dostępu zdalnego. W ramach każdego laboratorium uruchomione jest kilka takich samych kopii środowiska, aby każdy uczeń mógł pracować osobno.



Rysunek 1: Wysokopoziomowa koncepcja systemu

1.2.2. Koncepcja systemu

Na system będą składać się moduły pozwalające skonfigurować laboratorium z wstępnie przygotowanym zadaniem, udostępnić je uczniom oraz zebrać wyniki ich prac. Wysokopoziomowa koncepcja systemu została graficznie przedstawiona na rysunku 1.

Konfiguracja laboratorium będzie odbywać się za pomocą prostego interfejsu, w którym będzie można wprowadzić informacje, takie jak: nazwa, okres udostępniania, ilość maszyn oraz rodzaj środowiska. Po wprowadzeniu poprawnych danych system prześle konfigurację do modułu odpowiadającego za jego utworzenie. Proces tworzenia powinien być jak najbardziej uproszczony, tak aby osoby, które nie posiadają dużego doświadczenia, były w stanie skonfigurować laboratorium.

Po utworzeniu, będzie możliwe uruchomienie wielu instancji laboratorium, które zostaną uruchomione na jednej z maszyn hostujących. Parametry dostępne do maszyny zostaną zapisane oraz udostępnione dla użytkowników. Będą oni mogli połączyć się do nich za pomocą np. protokołów pulpitu zdalnego oraz wykonać przygotowane zadanie.

Ostatni z modułów pozwoli nauczycielowi na pobranie wszystkich plików z laboratorium znajdujących się w wyznaczonym folderze. Otrzyma je jako archiwum i będzie ono dotyczyć wszystkich instancji stworzonych dla wybranego laboratorium.

1.2.3. Sieci programowalne

Sieci programowalne są bardzo szeroko pojętym zagadnieniem. Rozumiemy przez nie technologie pozwalające nam operować od warstwy 2. modelu OSI, tworząc własne protokoły, przez warstwę 3 i zarządzanie na poziomie pakietów, po warstwy wyższe. Przykładowe, gotowe laboratorium, dostępne w systemie będzie przedstawiało to zagadnienie w oparciu o język P4 [5], który pozwala na sterowanie pakietami. Inne laboratorium będzie również oparte o framework Kathara [4] który pozwala na wirtualizację sieci złożonej z kilku urządzeń na jednej maszynie.

1.3. Istniejące rozwiązania

Za sprawą przejścia w ostatnim czasie na model zdalny w nauczaniu słowo “laboratorium” zmieniło trochę znaczenie. Do tej pory kojarzyło się ono głównie z salą zawierającą wiele komputerów. Zdarzało się nawet, że proces organizacji nowego laboratorium odbywał się w zupełnie nieautomatyzowany sposób i wymagał zainstalowania oprogramowania na każdym z stanowisk.

Aktualnie często używanym modelem jest stworzenie obrazu maszyny wirtualnej i udostępnienie go uczniom/studentom. Wymaga on jednak wiedzy oraz odpowiedniego sprzętu ze strony użytkowników. Szczególnie na niższych szczeblach edukacji nie każdy uczeń posiada wiedzę z zakresu uruchamiania maszyn wirtualnych, a często również nie ma wystarczająco wydajnego komputera, który to umożliwia. Inną możliwością jest także zainstalowanie lokalnie potrzebnego oprogramowania, co często również jest problematyczne i niepotrzebnie zabiera czas wielu osobom.

Przedstawiona aplikacja ma na celu ułatwić oba wspomniane wyżej rozwiązania i rozwiązać problemy standardowego podejścia do tego zagadnienia. Zarówno w modelu zdalnym jak i stacjonarnym może ona zastąpić istniejące rozwiązania.

Istnieją rozwiązania zapewniające wirtualne środowisko na żądanie, takie jak: Automat-IT [1] lub Wipro [31]. Są to jednak produkty komercyjne, nastawione na użytkownika biznesowego, który chce udostępniać w ten sposób tworzoną aplikację w celu na przykład przetestowania jej przez testerów. Wymagania systemu edukacji są bardzo często znacznie mniejsze oraz mają inną charakterystykę.

Laboratoria z każdej dziedziny mogą zostać zrealizowane w przygotowanym systemie. Jako laboratorium przykładowe dostępne będzie laboratorium z sieci programowalnych oparte o framework Kathara [4]. To rozwiązanie zastąpiło popularnego Netkit’a, poprzez implementację opartą o popularną technologię, jaką jest Docker [8]. Innymi rozwiązaniami do symulowania sieci są również GNS3 [15] oraz Cisco Packet Tracker [6]. Są one jednak rozwiązaniami mniej ogólnymi. Niemniej jednak bardzo łatwo mogą zostać przygotowane szablony laboratoriów z tymi produktami.

1.4. Analiza zagrożeń

Laboratoria wymagają utworzenia maszyny wirtualnej dla każdego ucznia, co generuje bardzo dużą ich liczbę. Utworzenie dziesiątek maszyn na jednym hoście może być niemożliwe. Aby temu zapobiec można podzielić ich uruchamianie pomiędzy wiele hostów. Wymaga to jednak zaprojektowanie skalowalnej struktury aplikacji.

Inną bardzo ważną rzeczą, o którą należy zadbać tworząc aplikację jest bezpieczeństwo. Ze względu na to, że laboratoria będą uruchomione na zdalnych hostach, a aplikacja będzie komunikować się z systemem, będzie ona narażona na wiele podatności, które mogłyby zostać wykorzystane przez potencjalnych atakujących.

Ze względu na ciągły rozwój sieci programowalnych oraz brak stabilnych rozwiązań, ciężko stwierdzić skuteczność sugerowanych narzędzi. W przypadku problemów z tworzeniem laboratorium sieci wirtualnych można wykorzystać inną dostępną technologię sieciową i przygotować laboratorium do pracy z ugruntowaną technologią, taką jak na przykład REST.

Innym zagrożeniem może być wsparcie wszystkich dostawców maszyn wirtualnych. Vagrant zapewnia wsparcie dla większości z nich, jednak może okazać się, że każdy wymaga odrębnej konfiguracji. W razie problemów można wesprzeć tylko jednego lub kilku wybranych z nich.

Tabela 1 przedstawia podsumowanie zagrożeń.

Ryzyko	Ważność	Minimalizacja ryzyka	Akcje naprawcze
zbyt duża ilość maszyn przypadająca na hosta	WYS.	- podział zadań tworzenia maszyn pomiędzy wiele hostów - skalowalna struktura kodu i aplikacji	- zwiększenie ilości hostów - zwiększenie mocy hostów - lazy loading (ładowanie maszyn dopiero gdy są one potrzebne)
atak zewnętrzny	WYS.	- stosowanie dobrych praktyk tworzenia aplikacji - walidacja	- naprawa wykrytych podatności
brak stabilności technologii sieci programowalnych	ŚRED.	- staranny dobór frameworka dla sieci programowalnych - wsparcie wielu technologii	- przygotowanie laboratorium dla stabilnej technologii jak np. REST
brak wsparcia dla różnych managerów maszyn wirtualnych	NISKA	- stosowanie narzędzi wspierających wszystkie technologie	- implementacja wsparcia tylko dla jednego supervisor'a

Tabela 1: Analiza zagrożeń

2. Zakres funkcjonalności

Rozdział przedstawia kontekst użytkowania produktu oraz specyfikuje wymagania funkcjonalne i нефunkcjonalne.

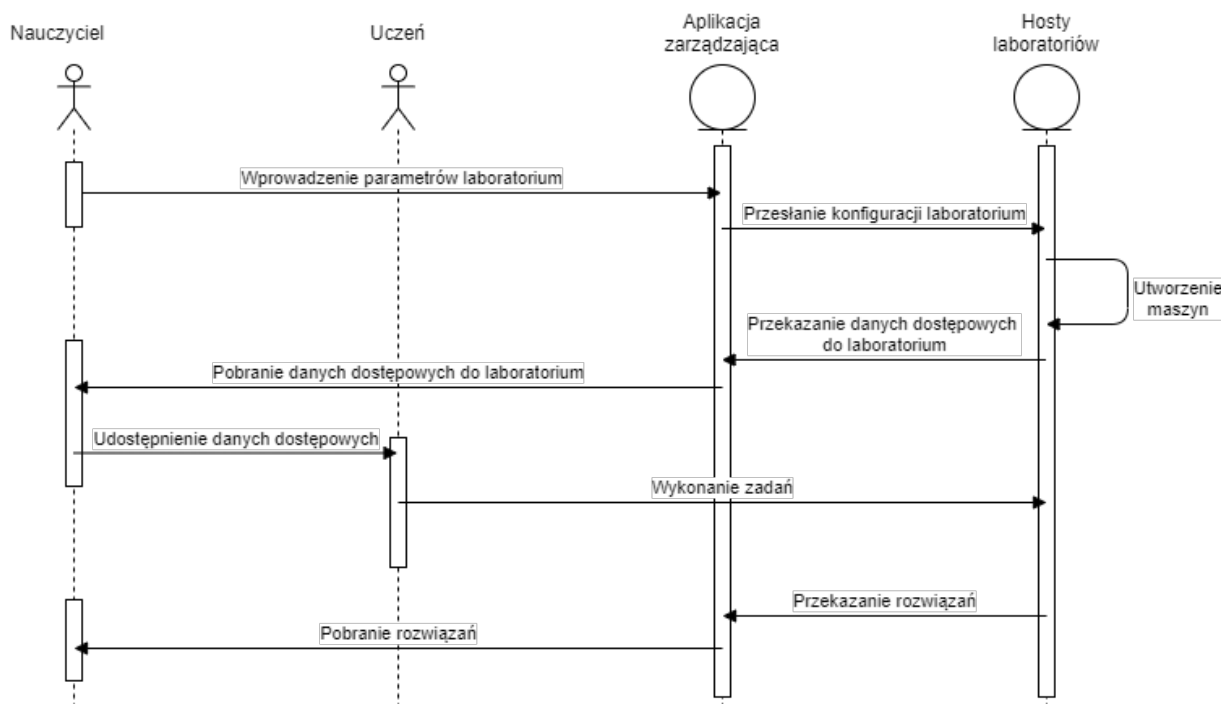
2.1. Kontekst użytkowania

W bazowym scenariuszu system może zostać wykorzystany do przygotowania laboratorium z dowolnego zakresu oraz do oceny przygotowanego zadania.

Aktorami wywołującymi akcje systemu będą:

- Administrator - jego zadaniem będzie uruchomienie i nadzór aplikacji w środowisku, do którego będą mieć dostęp nauczyciel i uczniowie, a także dbałość o ciągłe działanie systemu.
- Nauczyciel - będzie miał możliwość określenia parametrów laboratorium podczas konfiguracji. Następnie, po wykonanym przez uczniów zadaniu, będzie miał możliwość pobrania rozwiązań z wyznaczonego folderu znajdującego się w laboratorium.
- Uczeń - loguje się na stworzoną przez nauczyciela maszynę, poprzez otrzymane dane dostępu. Dzięki temu będzie mógł wykonać otrzymane zadanie w gotowym środowisku.

Diagram na rysunku 2 przedstawia kolejność podstawowych czynności wykonywanych w systemie.



Rysunek 2: Diagram sekwencji

2.2. Wymagania funkcjonalne

System powinien oferować następujące funkcjonalności:

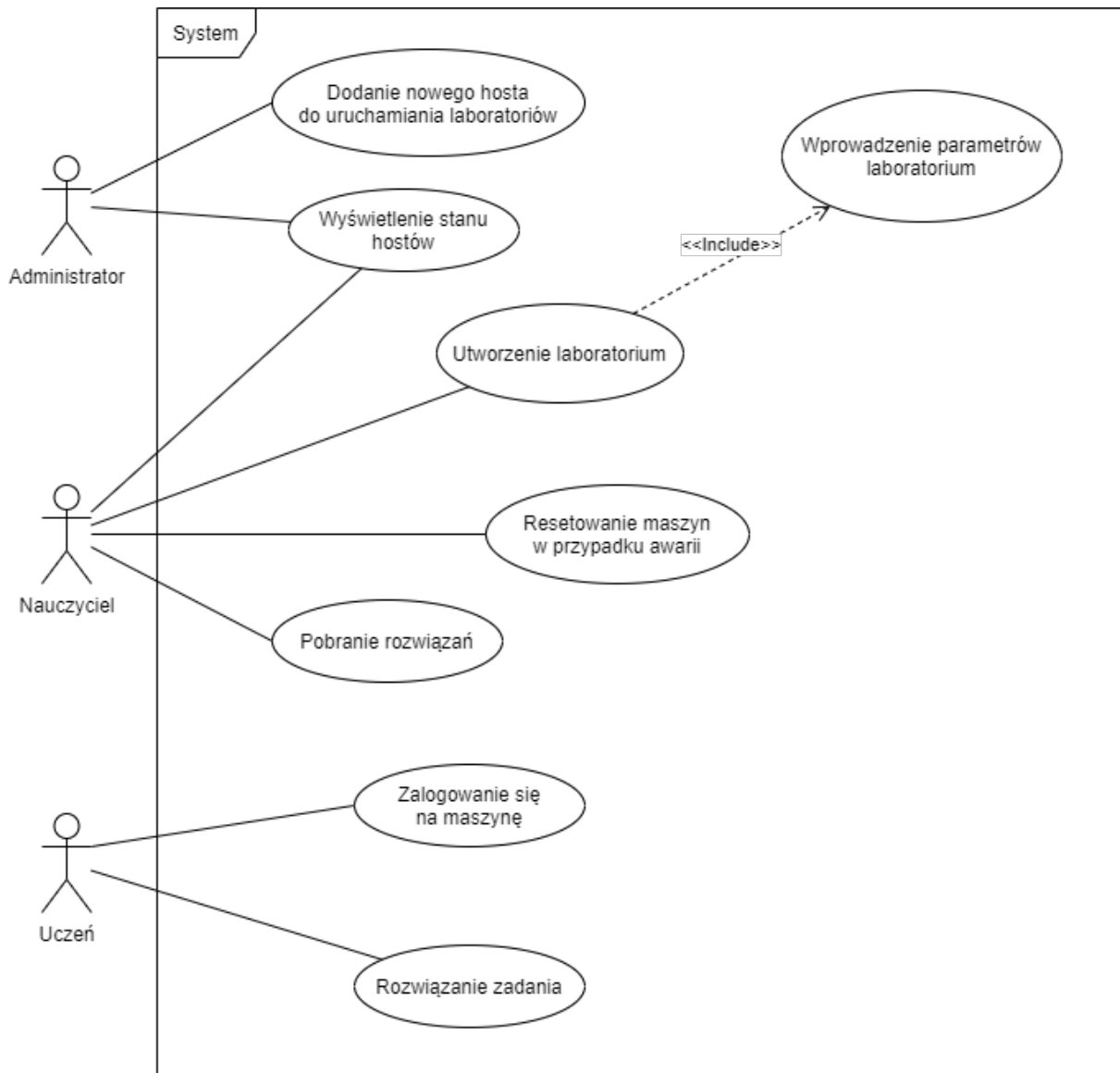
- Logowanie się do systemu dla użytkownika.
- Automatyczne tworzenie laboratoriów z wybranym środowiskiem do pracy:
 - Wpisanie nazwy,
 - Zdefiniowanie okresu udostępnienia.
- Uruchomienie wielu instancji danego laboratorium.
- Udostępnienie wcześniej przygotowanych szablonów do laboratoriów sieci programowalnych.
- Połączenie się z maszyną i wykonanie zadania.
- Zablokowanie dostępu do laboratorium po wyznaczonej dacie.
- Pobranie plików znajdujących się w ustalonym folderze.
- Restartowanie poszczególnych instancji w razie awarii.
- Wyświetlenie listy dostępnych hostów, na których można stworzyć laboratorium.

Przypadki użycia w formie graficznej przedstawione zostały na rysunku 3.

2.3. Wymagania нефunkcjonalne

System powinien spełniać następujące wymagania нефunkcjonalne:

- Udostępnienie nauczycielowi prostego interfejsu, którego celem będzie umożliwienie realizacji założeń.
- Udostępnienie laboratorium z zachowaniem określonych praw dostępu, w szczególności na określony czas.
- Otwartość na dalszy rozwój pod kątem użycia innych technologii oraz rozwoju poszczególnych modułów.
- Skalowalność systemu oraz łatwość dodawania nowych hostów, na których mogą być uruchamiane instancje laboratorium.



Rysunek 3: Diagram przypadków użycia

3. Wybrane aspekty realizacji

Niniejszy rozdział zawiera techniczny opis aplikacji. Jest w nim przedstawiona struktura i zasada działania systemu. Uwzględnione zostały także wykorzystane rozwiązania technologiczne, istotne mechanizmy i zastosowane algorytmy.

3.1. Koncept technologiczny

Środowisko laboratorium będzie udostępnione w postaci obrazów lekkich kontenerów. Do tworzenia obrazów zdecydowano się na wykorzystanie głównie narzędzia Docker, ze względu na jego popularność, a także pełną zgodność obrazów ze standardami innych narzędzi (zgodność ze standardem OCI [23]). Kontenery są stworzone, aby móc zapisywać konfigurację środowiska i odtwarzać ją wielokrotnie. Pozwalają one zrealizować wszystkie potrzebne operacje, jak maszyny wirtualne. Jednak potrzebują one mniejszej ilości zasobów, co wpływa korzystnie na możliwość korzystania z dostarczonego rozwiązania przez wiele użytkowników jednocześnie. Dodatkowym atutem tego podejścia jest fakt wyeliminowania ryzyka związanego z brakiem wsparcia dla różnych managerów maszyn wirtualnych. Docker pozwala także na uruchomienie kilku kontenerów z jednego obrazu co pozwoli zrealizować wymaganie wielu instancji. Aplikacja wspiera jednak również laboratoria przygotowane przy pomocy maszyn wirtualnych.

Technologia, która użyta do napisania systemu, musi umożliwiać komunikację z system operacyjnym w prosty sposób. Z tego powodu zdecydowano się na użycie języka Python, który jest bardzo popularny, co implikuje dużą ilość rozwiązań w postaci bibliotek.

Opis struktury oraz zarządzanie konkretnym laboratorium odbywa się za pomocą narzędzia Vagrant. Jest to narzędzie pozwalające na tworzenie i konfigurowanie maszyn wirtualnych. Obsługuje on większość systemów nadzorców maszyn wirtualnych, a także kontenery. Możemy dzięki niemu, tworząc opis maszyny, dostarczyć wszystkim użytkownikom w pełni skonfigurowane środowisko zawierające wszystko, czego potrzebują do pracy.

Przykładowe laboratorium będzie zawierać w pełni skonfigurowany framework Kathara, który jest implementacją środowiska Netkit. Netkit umożliwia emulację skomplikowanych infrastruktur sieciowych na jednej maszynie. Kathara implementuje to środowisko w oparciu o kontenery Docker'a oraz język Python. Zawiera on już wstępnie wiele technologii, takich jak na przykład P4, OpenVSwitch, Quagga, FRRouting, Bind. Z tego powodu jest to dobry wybór dla bazowej maszyny, która w założeniu ma udostępniać możliwość pracy z sieciami programowalnymi.

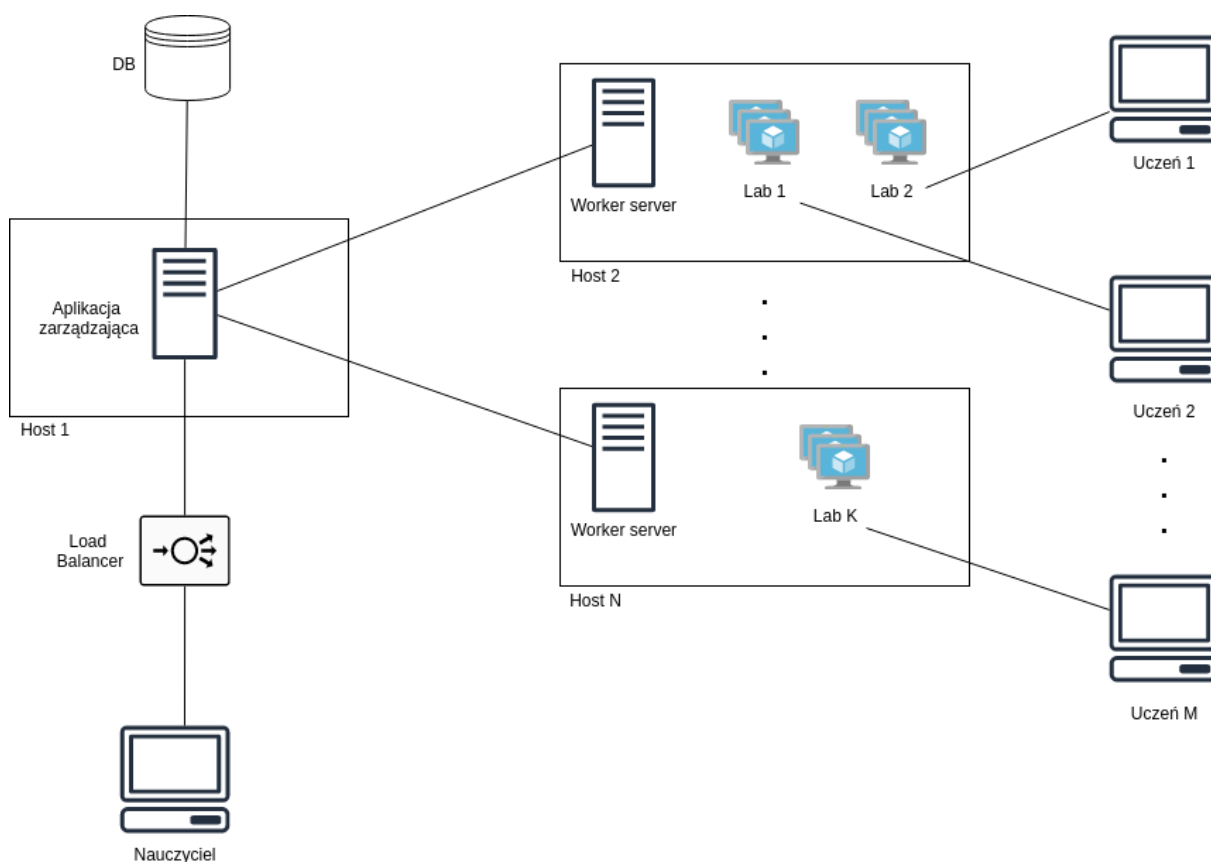
Interfejs do zarządzania laboratoriami jest aplikacją webową. Jest to aktualnie najpopularniejsze rozwiązanie, ze względu na łatwość dostępu. Do napisania aplikacji wykorzystany zostanie framework React.

3.2. Architektura systemu

Uruchamianie wielu maszyn wirtualnych wymaga wielu zasobów. Zatem cały system musi być wysoce skalowalny i łatwo rozszerzalny o nowe fizyczne maszyny, zapewniające dodatkową moc obliczeniową. Aby spełnić te wymagania system został podzielony na mniejsze komponenty. Każdy z nich ma określone odpowiedzialności, co pozwala na wygodną pracę nad wybranym elementem systemu.

- workerzy - zarządcy poszczególnych hostów, umożliwiając utworzenie i uruchomienie maszyn wirtualnych z laboratoriami. Następnie mają one możliwość odczytania konfiguracji i zwrócenie jej do głównego serwera
- aplikacja zarządzająca - koordynuje pracę mniejszych serwerów nazwanych workerami. Tworzy to architekturę opartą o mikroserwisy.
- interfejs systemu - aplikacja webowa, dzięki której użytkownik komunikuje się z systemem

Kolejne hosty mogące uruchamiać laboratoria są dodawane do systemu poprzez uruchomienie na nich programu workera. Nie ma górnego ograniczenia na ich ilość, co zapewnia łatwą skalowalność.



Rysunek 4: Diagram architektury systemu

3.3. Komponenty systemu

W poniższej sekcji znajdują się opisy poszczególnych części systemu, które zostały przedstawione na rysunku 4. Zawierają opisy użytych technologii oraz przedstawiają dokładne działanie i komunikację pomiędzy sobą.

3.3.1. Szablony laboratoriów

Większość laboratoriów jest zrealizowana za pomocą kontenerów dockerowych. Środowisko laboratoriów jest opisywane za pomocą plików Dockerfile [10]. Jest tam tworzony obraz kontenera oraz instalowane potrzebne oprogramowanie. Taki plik tworzy szablon laboratorium z którego później powstają jego instancje.

Część potrzebnego oprogramowania jest taka sama dla wszystkich typów laboratoriów. Został więc przygotowany obraz bazowy, który może być wykorzystany do utworzenia kolejnych szablonów. Bazuje on na kontenerze Ubuntu i zawiera zainstalowane serwery *rdp* oraz *ssh*, umożliwiające zdalny dostęp.

Z szablonu jest tworzone laboratorium, w ramach którego może działać wiele instancji obrazu. Aby nimi zarządzać wykorzystany został program Vagrant [30]. Zapewnia on interfejs sterowania kontenerami oraz zajmuje się przydzielaniem portów.

Każda z instancji obrazu kontenera w ramach laboratorium ma także podlinkowane pliki z katalogu `/home/lab` do maszyny na której są uruchomione, tak aby dane z tego folderu pozostały niezmienione w przypadku restartu lub wyłączenia instancji.

Przykładowy szablon został załączony w załączniku A.

Powstały następujące szablony laboratoriów:

- Laboratorium bazowe - zawiera serwery rdp oraz ssh. W oparciu o niego powinny być tworzone następne szablony.
- Kathara - bazuje na powyższym, a także zawiera zainstalowane programy Docker oraz Kathara. Zawiera również sklonowane przykładowe zadania dostępne w repozytorium frameworka Kathara. [19]
- RStudio - szablon środowiska zawierającego program RStudio, który ułatwia pracę z językiem R używanym w statystyce. Został on przygotowany w celu pokazania, że system łatwo można rozszerzyć o nowe laboratoria
- P4 - szablon maszyny stosowanej do tutoriali ze strony P4lang. Używa on VirtualBox zamiast Dockera.

Każdy szablon poza plikami niezbędnymi do utworzenia instancji, zawiera również metadane, takie jak nazwa oraz jaki zarządca jest potrzebny do uruchomienia. Aktualnie występują dwa typy zarządców - Docker oraz VirtualBox. Dane te przechowywane są w pliku

lab_template.json. Aplikacja używa tych informacji do wybrania i uruchomienia laboratorium.

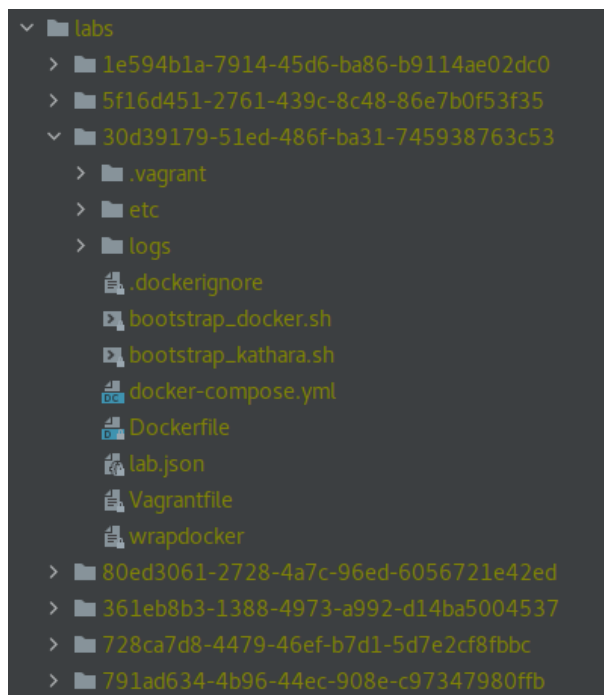
Wszystkie szablony przechowywane są w osobnym repozytorium, które jest niezależne od repozytorium głównej aplikacji. Wspomaga to osobne rozwijanie szablonów oraz ich aktualizację w czasie działania programu.

3.3.2. Worker

Worker jest programem pozwalającym wykonywać operację na laboratoriach na jednym hoście i udostępniającym swój interfejs za pomocą REST API. Każde laboratorium jest przypisywane do dokładnie jednego workera.

Przechowywanie danych laboratoriów

Workery przechowują dane w jako pliki, nie używając żadnej bazy danych. Jak przedstawiono na rysunku 5 w konfiguracji definiowany jest główny folder laboratoriów, w którym podfoldery oznaczają osobne laboratoria oraz nazwane są za pomocą ich identyfikatorów. Na tej podstawie możliwe jest znalezienie wszystkich laboratoriów oraz ich uruchomienie.



Rysunek 5: Struktura folderu przechowująca laboratoria

Każdy z nich zawiera konfigurację przechowywaną w pliku 'lab.json', dane oraz pliki konfiguracyjne kontenerów (Dockerfile, Vagrantfile oraz skrypty).

Komunikacja z aplikacją zarządzającą

Każdy z worker'ów po uruchomieniu rejestruje się w aplikacji zarządzającej jako dostępny. Następnie ponawia rejestrację co 10 sekund. Każda z akcji rejestracji zawiera w sobie skonfigurowane przed uruchomieniem informacje, takie jak adres IP oraz port, pod którym jest widoczny

z zewnątrz. Posiada także unikalny identyfikator, który nie może ulec zmianie, ponieważ wtedy aplikacja zarządzająca rozpozna worker'a jako inny program i nie powiąże z nim już utworzonych wcześniej laboratoriów. Jeśli aplikacja zarządzająca nie otrzyma przez 30 sekund pakietu rejestracji, oznacza ona wtedy worker'a jako niedostępnego i nie tworzy na nim nowych laboratoriów.

Dostępność z zewnątrz

Worker uruchamia laboratoria na hoście na którym się znajduje. Aby połączyć się więc do laboratorium uczeń musi mieć możliwość dostępu do hosta. Z tego powodu musi być udostępniony pod adresem publicznym. Problematiczne może być, jeśli po drodze dochodzi do translacji adresów przy użyciu PAT, ponieważ każda instancja laboratorium musi być dostępna na porcie, który jest przypisywany dynamicznie. W takiej sytuacji należałoby dla każdego laboratorium przekierowywać porty na routerze dokonującym translacji lub w firewallu. Rozwiązaniem niewymagającym dodatkowej pracy jest dostępność zakresu portów z zewnątrz. Może to jednak stanowić zagrożenie dla bezpieczeństwa.

Włączanie i wyłączenie laboratoriów w określonym czasie

Program uruchamia także powtarzające się zadania, które odpowiadają za włączanie i wyłączanie laboratoriów w określonym czasie. Są to zadania działające w różnym czasie. Po uruchomieniu się zadania, sprawdzają one przygotowane/działające laboratoria i włączają lub wyłączają je, aby dostosować się do wprowadzonego przez użytkownika przedziału dat. Dokładność uruchomienia laboratorium wynosi około 2 minut, natomiast dokładność jego wygaśnięcia to około 5 minut.

Sterowanie laboratoriami

Włączanie i wyłączanie całych laboratoriów lub instancji poszczególnych maszyn odbywa się za pomocą komend kierowanych do systemu. Są to głównie komendy programu Vagrant wykonywane z odpowiednimi argumentami. Dzięki temu szablony laboratoriów mogą być tworzone w oparciu o dowolnego zarządcę maszyn wspieranego przez narzędzie Vagrant. Ze względu na wykonywanie komend bezpośrednio na systemie jest to jeden z bardziej niebezpiecznych punktów systemu i jest wymagana odpowiednia walidacja komend.

Zagnieżdżona wirtualizacja

Nietrudno zauważyć, że w systemie dochodzi do uruchamiania zagnieżdżonych kontenerów. Aplikacja worker'a działa w kontenerze, w którym uruchamiane są laboratoria w postaci kontenerów. Laboratoria mogą zawierać framework Kathara, który symuluje urządzenia sieciowe również za pomocą kontenerów.

Nie jest to trywialne zadanie i wymaga dodatkowej konfiguracji. Aby było to możliwe uruchamia się kontenery w trybie uprzywilejowanym oraz linkuje folder `/var/lib/docker` z kontenera do hosta. Docker działający więc w kontenerze nie jest pełnoprawną osobną instancją, a korzysta ze swojego odpowiednika na hoście. Sposób ten został dokładnie opisany w artykule [26].

Uruchamianie uprzywilejowanych kontenerów może mieć negatywny wpływ na bezpieczeństwo aplikacji. Po zaimplementowaniu opisanego wcześniej sposobu, został on oznaczony jako przestarzały przez wzgląd na bezpieczeństwo. Powinien zostać on zastąpiony przez tzw. “root-less containers”, które nie wymagają trybu uprzywilejowanego.

API

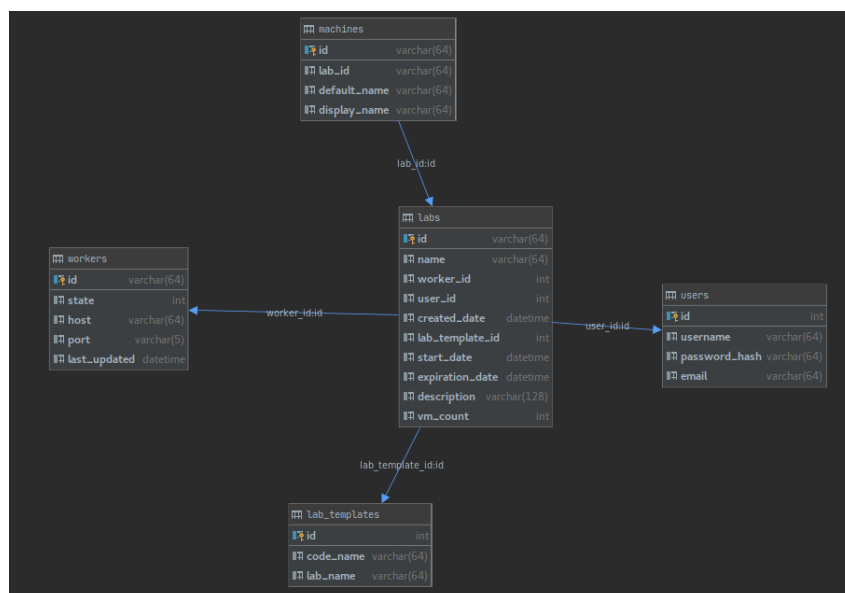
Do udostępnienia REST API workera został użyty framework Flask wraz z biblioteką Flask-RESTPlus [12]. Za pomocą zapytań HTTP można wykonać następujące operacje:

- Utworzenie laboratorium
- Pobranie status oraz informacji o laboratorium
- Pobranie logów z danego laboratorium
- Restart maszyny w obrębie laboratorium
- Pobranie plików z laboratorium w postaci archiwum zip

3.3.3. Aplikacja zarządzająca

Jest to aplikacja, która jest pośrednikiem pomiędzy użytkownikami tworzącymi laboratoria, a poszczególnymi workerami. Została ona podzielona na 3 podkomponenty.

Baza danych



Rysunek 6: Diagram bazy danych

Ze względu na małą złożoność danych została wykorzystana baza danych oparta na pojedynczym pliku z wykorzystaniem SQLite. Struktura kodu pozwala na zmianę systemu

bazodanowego i w prosty sposób uwzględnienie jej w istniejącym rozwiązaniu.

Struktura bazy została przedstawiona na rysunku 6. W bazie danych są przechowywane informacje, takie jak:

- laboratoria - metadane o laboratorium, takie jak: użytkownik, który je utworzył, host, na którym działa, data, typ
- hosty (workery) - informacje o uruchomionych workerach, np. aktualny stan, data ostatniej aktualizacji stanu
- użytkownicy - dane o użytkownikach mających dostęp do aplikacji, np. login, hash hasła
- maszyny - powiązania nazw wprowadzonych przez użytkownika z nazwami kodowymi maszyn w obrębie laboratorium
- szablony - obsługiwane szablony laboratoriów (zgodne z informacjami zawartymi w repozytorium szablonów)

Backend

Jest to aplikacja napisana w języku Python wykorzystująca REST API. Stanowi łącznik pomiędzy frontendem, a workerami, ukrywając przed użytkownikiem fakt rozproszenia systemu. Został wykorzystany framework Flask, który pozwolił na udostępnienie interfejsu aplikacji w prosty sposób.

Operacje udostępnione przez backend można podzielić na następujące grupy:

- Uwierzytelnianie i autoryzacja - weryfikacja danych podanych przez użytkownika przy logowaniu, generowanie tokenu do autoryzacji
- Workerzy - udostępnienie listy workerów wraz ze statusami
- Laboratoria - pobieranie listy laboratoriów dla danego użytkownika, tworzenie nowego laboratorium - zlecenie wykonania tego zadania do wybranego, dostępnego workera na podstawie algorytmu Round-Robin, pobieranie szczegółów lub plików wybranego laboratorium, restartowanie wybranej maszyny dla danego laboratorium
- Dostępne typy laboratoriów - endpoint, który zwraca wszystkie dostępne szablony laboratoriów

Backend bazuje na wartościach zawartych w bazie danych - użytkownicy, laboratoria, workerzy.

Do udostępnienia REST API zgodnego ze standardami wykorzystano rozszerzenie frameworka Flask - Flask-RESTPlus. W łatwy sposób pozwoliło to na zdefiniowanie modeli wykorzystywanych przez poszczególne endpointy, dodanie walidacji, przekazywanie odpowiednich wyjątków, jak na przykład HTTP 401 - Unauthorized. Ułatwiło to także skorzystanie

z narzędzia Swagger, który prezentuje w przejrzysty sposób interfejs aplikacji.

Inną odpowiedzialnością backendu jest także zarządzanie nazwami maszyn w obrębie laboratorium. Aby użytkownik mógł łatwo je rozróżnić, wprowadza on własne aliasy podczas tworzenia, które następnie backend przechowuje w bazie danych oraz podmienia w momencie wysyłania danych do użytkownika.

Frontend

Frontend stanowi prosty interfejs webowy dla użytkownika, który umożliwia tworzenie nowych oraz przegląd istniejących już laboratoriów oraz ich szczegółów, a także przegląd dostępnych workerów. Do jego stworzenia został wykorzystany język TypeScript w połączeniu z frameworkiem React. Technologia ta została wybrana ze względu na:

- wydajność - React jest uznawany za jedną z najwydajniejszych technologii, które są aktualnie popularne,
- chęć poznania - żaden z członków zespołu nie miał większego doświadczenia z użytymi technologiami webowymi, co stanowiło dobrą okazję do zapoznania się z popularnymi narzędziami,
- niski próg wejścia - popularność i ilość dostępnych materiałów pozwoliła na szybkie rozpoczęcie prac nad interfejsem użytkownika.

Aby wygląd aplikacji był estetyczny i zgodny z aktualnymi standardami wykorzystano komponenty z biblioteki React Bootstrap. Dodatkowo zadbane o wygodę oraz lepsze odczucia z użytkowania aplikacji poprzez dodanie między innymi “date pickera” oraz wskaźnika postępu podczas ładowania danych z backendu. Architektura, jak i sama aplikacja, nie jest skomplikowana. Ze względu na małą liczbę widoków nie było potrzeby stosowania bibliotek takich jak na przykład Redux do zarządzania stanem. Wszystkie dane na bieżąco pobierane są API, nie występuje więc potrzeba zapisywania danych lokalnie. Jediną rzeczą przechowywaną w lokalnym rekordzie opisującym stan jest token użytkownika.

3.3.4. Load Balancer

Do systemu został dołączony również program typu *load balancer*. W tym celu został użyty program *haproxy* [17]. Wykorzystywany jest on do udostępnienia aplikacji zarządcy pod jednym adresem, tak aby dostęp do poszczególnych komponentów odbywał się za pomocą ścieżek w adresie, a nie adresów i portów.

Umożliwia on także skalowalność aplikacji i dodawanie większej ilości instancji podkomponentów aplikacji zarządcy.

3.3.5. Uwierzytelnianie i autoryzacja

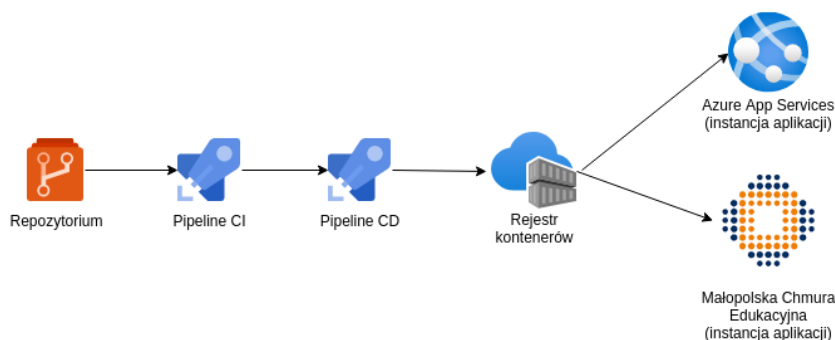
Ze względu na wymóg dostępu do systemu przez wielu użytkowników potrzebna była obsługa uwierzytelniania i autoryzacji. Każdy użytkownik posiada własne konto, które jest identyfikowane za pomocą nazwy użytkownika. Aby zalogować się do aplikacji potrzebne jest także

podanie hasła.

Kontami użytkowników zarządza tylko administrator. Jest on odpowiedzialny za tworzenie kont i ustawianie hasła. Aktualnie system nie wspiera zakładania kont przez aplikację, za pomocą opcji rejestracji.

Proces autoryzacji odbywa się za pomocą tokenów JWT. Po poprawnym zalogowaniu za pomocą nazwy użytkownika oraz hasła, klient otrzymuje token, który zapisuje, a następnie dołącza go do każdego wykonywanego zapytania do API. W tokenie znajduje się informacja o nazwie użytkownika oraz dacie wygaśnięcia tokenu. Rozwiązanie jest bezpieczne ze względu na to, że informacje te zostają zaszyfrowane kluczem prywatnym serwera, więc praktycznie nie jest możliwe odczytanie albo zmanipulowanie tokenu.

3.4. CI/CD



Rysunek 7: Diagram procesów CI/CD

W procesie wytwarzania oprogramowania ważną rolę odgrywają też procesy Continuous Integration (CI) oraz Continuous Delivery (CD). Zapewniają one, że wadliwy kod (niebudujący się lub nie przechodzący testów) nie może zostać wdrożony, ale także automatyczne wdrożenie najnowszej wersji dla użytkowników.

Do wdrożenia tych procesów zostało użyte narzędzie Azure DevOps [3], co zostało przedstawione na rysunku 7. Realizuje ono te procesy za pomocą tzw. pipeline'ów. Na potrzeby systemu zostały stworzone dwa pipeline'y. Jeden z nich zapewnia brak możliwości zintegrowania kodu z główną gałęzią, gdy nie przechodzą testów, drugi zapewnia automatyczne opakowanie aplikacji w obrazy kontenerów dockerowych i przesłanie ich do rejestru kontenerów znajdującego się na platformie Azure [2]. Zapewnia to łatwą dostępność najnowszej wersji systemu.

3.5. Środowiska programistyczne

Dobłą praktyką w trakcie tworzenia systemu jest skonfigurowanie wielu jego instancji, z których każda ma określony cel i parametry uruchomienia, a także przechowuje inne dane. Przygotowane zostały więc dwa środowiska:

- deweloperskie - służy ono do pracy z aplikacją, zawiera obszerne logi oraz umożliwia przebudowanie przy zmianach w kodzie. Użyte tutaj klucze prywatne nie są bezpieczne. Powinno ono być używane tylko do lokalnej pracy.
- produkcyjne - instancja systemu dostępna dla użytkowników. Powinna być ona w pełni bezpieczna.

Oba te środowiska zostały zrealizowane za pomocą narzędzia Docker-Compose [9]. Umożliwia ono zdefiniowanie w pliku konfiguracyjnym uruchamianych kontenerów oraz parametrów z jakimi są one uruchamiane. Wtedy wystarczy tylko jedna komenda, aby wszystkie potrzebne komponenty zostały uruchomione.

3.6. Chmura obliczeniowa

Chmury nie bez powodów zdobywają teraz dużą popularność. Udostępniają one na żądanie bardzo wiele usług, które są kluczowe do działania aplikacji.

Za sprawą konteneryzacji wszystkich modułów, system jest łatwy do wdrożenia u dostawców chmurowych. Początkowo wybrane zostały usługi platformy Azure [2], ze względu na dostępność licencji dla studentów. Aby ułatwić konfigurację, zostało użyte narzędzie Terraform [28]. Jest to narzędzie pozwalające opisać infrastrukturę za pomocą kodu (Infrastructure as a Code). Architektura i zasoby z chmury są przez to łatwe do stworzenia i proces ten jest w pełni zautomatyzowany. Następnie aplikacja została przeniesiona na zasoby Małopolskiej Chmury Edukacyjnej [21]. Ze względu na skalowalność systemu programy worker mogą działać jednocześnie u wielu dostawców.

3.7. Szablon laboratorium P4

W pracy zawarty jest także temat sieci programowalnych. Kathara w swoich laboratoriach zawiera jedno zadanie realizujące ten zakres. Dodatkowo została podjęta próba przygotowania autorskiego laboratorium z zadaniem dotyczącym tego tematu również w oparciu o framework Kathara, skupiającego się na języku P4 [5]. Jest to język pozwalający w niskopoziomowy sposób sterować ruchem sieciowym.

W repozytorium obrazów Kathary dostępny jest także obraz “kathara/p4“, na którym bazuje wspomniane zadanie. Zawiera ono jednak prostą konfigurację, która nie była aktualizowana od około 3 lat. Po zapoznaniu się z nim okazuje się, że oprogramowanie tam zawarte jest przestarzałe, a sam obraz ma wielkość około 4GB. Jest to znacząco więcej niż przeciętny obraz użyty w systemie oraz ponad 10-krotnie więcej niż podobne obrazy udostępniane przez *P4lang* [25].

Możliwe rozwiązania tego problemu:

- Kathara pozwala również użyć własnych kontenerów. Aby przygotować to laboratorium, wskazane byłoby przygotowanie własnego obrazu zawierającego najnowsze oprogramo-

wanie. Przykładowa instalacja środowiska potrzebnego do uruchomienia programu P4 znajdują się w źródle [20].

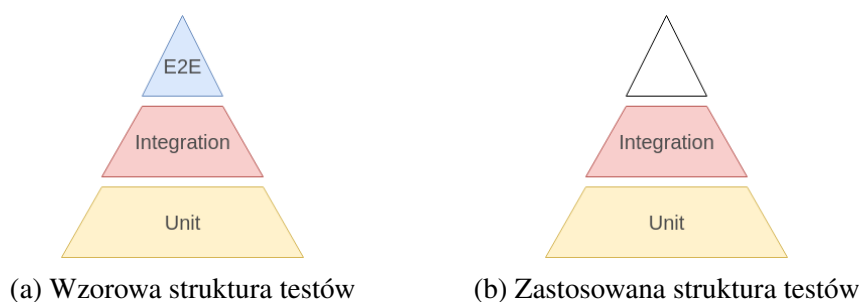
- P4lang [25] udostępnia wiele obrazów. Niektóre z nich zawierają potrzebne środowisko, jednak po testach okazuje się, że prawdopodobnie nie komunikują się poprawnie z pozostałymi hostami. Należałoby zweryfikować to podejście i wprowadzić ewentualne modyfikacje w celu stworzenia obrazu działającego poprawnie z frameworkiem.
- Użycie wspomnianego obrazu. Można stworzyć laboratorium pod wspieraną, lecz nieaktualną wersję języka, jednak może mieć ono znacznie mniejszą wartość edukacyjną.
- P4lang udostępnia także własne tutoriale oraz przygotowane dla nich środowisko w postaci pliku Vagrantfile.

W aktualnej wersji systemu zostało użyte ostatnie rozwiązanie, ze względu na łatwość wdrożenia. Obecne środowisko, wykorzystuje Netkit zamiast frameworka Kathara.

3.8. Testy aplikacji

Testy automatyczne są niezaprzeczalnie bardzo ważnym elementem każdej aplikacji. Pomagają one sprawdzić zgodność systemu z wymaganiami i wykryć błędy bez konieczności każdorazowej weryfikacji ręcznej. W opisywanym systemie również zostały napisane testy w celu podwyższenia jakości wytwarzanego oprogramowania.

Aktualnie dominującą architekturą testów jest struktura piramidy [13]. Jest to metafora sposobu myślenia o testach jako o warstwach o różnej granularności. Pokazuje ona też ilość testów na każdej warstwie. Przykładowa piramida przedstawiona została na rysunku 8a.



Rysunek 8: Piramida testów

Kolejne warstwy od dołu to:

- unit testy - testy pojedynczego fragmentu kodu, funkcji lub klasy. Stosują technikę *mockowania* pozostałych zależności. Powinny być najliczniejsze.
- testy integracyjne - inaczej serwisowe, testy większej części aplikacji integrujące więcej modułów. W przedstawionym systemie jest to np. pojedynczy moduł systemu. Również mockują pozostałe zewnętrzne zależności.

- testy e2e (end-to-end) - testują one działanie aplikacji od początku do końca dla realnych scenariuszy użytkowania. Najmniej liczna warstwa.

W opisywanym systemie została przygotowana architektura zawierająca testy z dolnej części piramidy, co zostało przedstawione na rysunku 8b. Dla modułów frontendu oraz backendu są to testy unit oraz integracyjne, dla workera natomiast tylko unit testy. Testy e2e ze względu na wysoki stopień skomplikowania spowodowany tworzeniem laboratorium będącym kontenerem/maszyną wirtualną zostały pominięte.

3.8.1. Backend oraz Worker

Oba te moduły zostały napisane z użyciem języka Python zostały więc zastosowane podobne narzędzia do testów. Do unit testów został użyty *Pytest* [27]. Na backendzie została również użyta technika *dependency injection* w kodzie aplikacji, aby ułatwić pisanie testów.

Dodatkowo na backendzie powstała również warstwa testów integracyjnych. Mockuje ona zewnętrzne zależności takie jak baza danych oraz worker. Frameworkiem do testów jest w tym przypadku również *Pytest*. Dla każdego uruchomienia testów tworzona jest nowa baza danych oraz startowana aplikacja w tle. Do zamockowania zewnętrznych zapytań HTTP zostało użyte narzędzie *Wiremock* [32]. Testy komunikują się z aplikacją za pomocą jej interfejsu REST, nie natomiast przez bezpośrednie wywoływanie funkcji.

3.8.2. Frontend

Taka sama struktura testów została zastosowana dla frontendu. Unit testy zostały napisane przy pomocy narzędzia *Jest* [18].

Do testów integracyjnych (z zamockowanym backendem) wykorzystany został framework *Cypress* [7]. W tym przypadku wymagane jest uruchomienie ręczne aplikacji, a potem testów które wykonują akcje w rzeczywistym środowisku przeglądarki.

4. Organizacja pracy

Rozdział zawiera informacje na temat organizacji pracy przy projekcie, narzędzi wykorzystywanych do jego stworzenia, członkach zespołu i podziale prac.

4.1. Planowanie i przebieg pracy

Od samego początku została przyjęta modułowa architektura systemu, co pozwoliło nam pracować równolegle nad różnymi częściami systemu. W pierwszej fazie, do przygotowania prototypu, praca nad projektem odbywała się w różnych trybach. Pozwoliło to zbadać aktualne możliwości, znaleźć dostępne narzędzia i przygotować działający prototyp systemu.

W następnych fazach projektu została przyjęta metodyka pracy, która opierała się na technikach *Agile*. Wszystkie zadania, które były już jasno określone i zdefiniowane, zostały podzielone na fazy, z których każda trwała tydzień. W ramach jednej fazy na każdego członka zespołu przypadało jedno większe zadanie, które realizował.

Często stosowaną praktyką był również *pair programming*, w ramach którego obaj członkowie zespołu pracowali nad jednym zadaniem. Ze względu na aktualną sytuację pandemiczną odbywał się głównie w formie zdalnej za pomocą narzędzia Microsoft Teams [22], dzięki któremu jedna osoba mogła udostępniać aktualny stan prac i obie osoby mogły prowadzić dyskusje na temat rozwiązania.

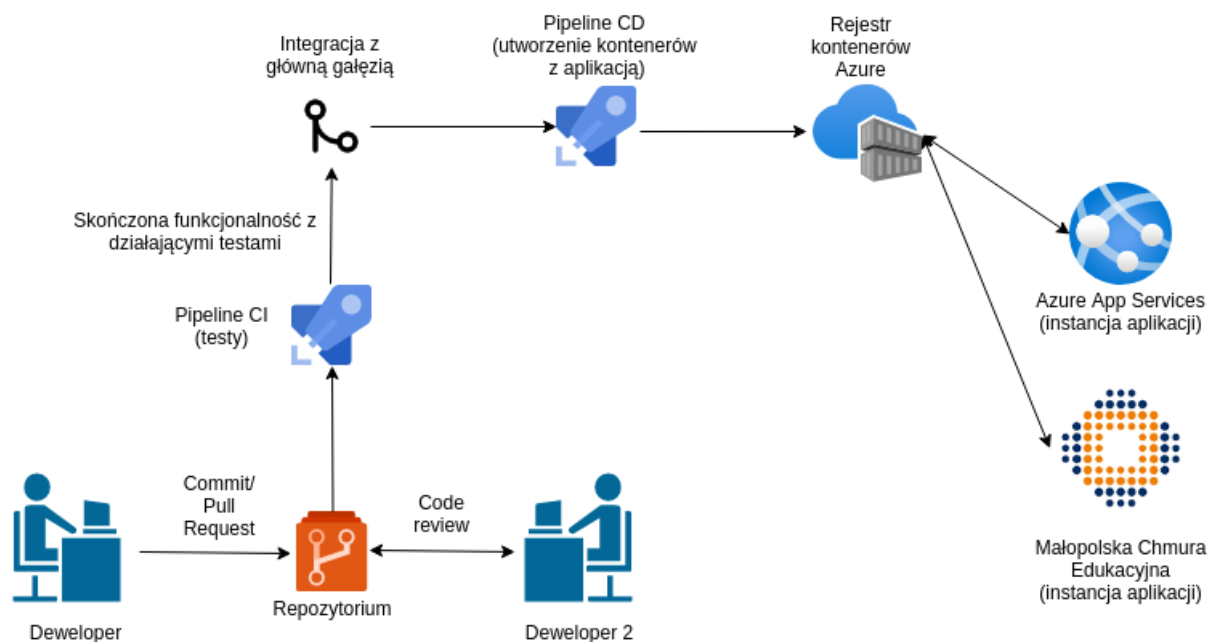
Do organizacji zadań została wykorzystana aplikacja Trello [29]. Jest to narzędzie pozwalające podzielić pracę zgodnie z założeniami tablicy w metodyce *Kanban*. W naszym przypadku wyodrębnione zostały 3 sekcje: “Do zrobienia”, “W trakcie”, “Zrobione”. Z powodu małego zespołu oraz faktu, że projekt nie jest bardzo dużym systemem, narzędzie to w pełni spełniło swoją rolę.

Po zrealizowaniu kilku faz (zwykle 3-4) następowała konsultacja z Opiekunem pracy w celu zaprezentowania efektów oraz zebrania uwag i poprawek do wprowadzania w następnych fazach. Spotkania odbywały się za pomocą platformy Google Meet/Hangouts [16]

Przygotowywanie dokumentacji odbywało się za pomocą narzędzia Overleaf [24], ze względu na możliwość równoległej pracy nad dokumentem. Do stworzenia diagramów zawartych w dokumentacji użyta została aplikacja Draw.io [11]

4.2. Techniczna organizacja pracy

Do zarządzania wersjami kodu użyliśmy narzędzia *git*, natomiast główne repozytorium znajdowało się najpierw na platformie GitLab [14], a następnie zostało ono zmigrowane na platformę AzureDevOps [3]. Zdecydowaliśmy się na zmianę narzędzie ze względu na lepszą integrację z chmurą Azure.



Rysunek 9: Diagram procesu wytwarzania kodu

W celu utrzymania jak najlepszej jakości kodu zostały przyjęte dosyć restrykcyjne założenia dotyczące procesu powstawania kodu. Każda gałąź pracy, najczęściej odwzorowująca jedno zadanie, mogła być dołączona do głównej gałęzi dopiero po przeprowadzeniu przez drugą osobę tzw. “code review” i rozwiązaniu wszystkich dyskusji oraz gdy pipeline CI zakończył się pomyślnie, czyli wszystkie testy wykonały się poprawnie. Nikt również nie mógł dodawać kodu bezpośrednio do głównej gałęzi. Po połączeniu gotowego zadania do głównej gałęzi automatycznie były tworzone kontenery w poszczególnych jej modułach oraz przesyłane do prywatnego rejestru kontenerów na platformie Azure. Pozwalało to szybko i łatwo uruchomić nową wersję aplikacji na dowolnym hoście. Poglądowy przebieg prac został przedstawiony na rysunku 9.

4.3. Osoby zaangażowane w projekcie

dr. inż. Sławomir Zieliński

Klient produktu - określenie oczekiwanego rezultatu projektu, a także osoba nadzorująca przebieg działań - opiekun pracy.

Arkadiusz Kraus i Mateusz Naróg

Studenci wydziału Informatyki, Elektroniki i Telekomunikacji Akademii Górniczo-Hutniczej realizujący projekt.

4.4. Podział zadań

Jednoznaczny wskazanie osoby odpowiedzialnej za wybrane zadania jest trudny do zrealizowania, ze względu na często stosowany “pair programming”. Dodatkowo starano się realizować kolejne funkcjonalności kompleksowo, więc jednocześnie pracowano nad wybraną kwestią za-

równy na frontendzie, backendzie, jak i workerze. Pod tym względem wynikł pewien naturalny podział obowiązków w zespole, więc możliwe jest wskazanie osób, które były w większości odpowiedzialne za dany moduł:

- Arkadiusz Kraus - Worker w połączeniu z bazowym szablonem laboratoriów
- Mateusz Naróg - Frontend
- Działania wspólne - Backend, testy aplikacji oraz CI/CD

4.5. Zrealizowane etapy pracy

Faza wstępna

03.2020 - 05.2020

Zapoznanie z tematem pracy, przegląd dostępnych narzędzi. Planowanie dalszych działań związanych z projektem. Przygotowanie prostego prototypu do tworzenia maszyny.

Przygotowanie architektury

06.2020 - 09.2020

Wykonanie podziału aplikacji na poszczególne moduły. Prace nad dockeryzacją oraz automatyzacją - CI/CD.

Aplikacja końcowa

10.2020 - 12.2020

Dodanie do aplikacji niezbędnych funkcjonalności.

Szablony laboratoriów, faza końcowa

11.2020 - 12.2020

Przygotowanie przykładowych laboratoriów. Wnoszenie poprawek do aplikacji.

Dokumentacja

03.2020 - 12.2020

Prace związane z dokumentacją pracy inżynierskiej.

Na rysunku 10 został przedstawiony harmonogram realizacji prac projektowych z dokładniejszym przedstawieniem wykonywanych zadań podczas poszczególnych etapów pracy.

Etapy pracy	Szczegóły	03-05	06-09	1.10-20.10	21.10-27.10	28.10-03.11	4.11-10.11	11.11-17.11	18.11-24.11	25.11-01.12	02.12-08.12	09.12-15.12	16.12-22.12
Faza wstępna	Zapoznanie z tematem												
	Przegląd dostępnych narzędzi												
	Planowanie działań												
	Prosty prototyp do tworzenia maszyny												
Przygotowanie architektury	Organizacja struktury projektu												
	Dockeryzacja z uwzględnieniem środowisk programistycznych												
	Przygotowanie Azure Pipelines												
	Dodanie testów												
Aplikacja końcowa	Przygotowanie bazowego interfejsu użytkownika												
	Dodanie obsługi parametrów przy tworzeniu laboratoriów												
	Pobieranie statusu oraz danych dostępowych dla wybranego laboratorium												
	Obsługa zatrzymywania oraz restartowania maszyn												
	Pobieranie plików z wybranego laboratorium												
	Poprawienie user experience na frontendzie												
	Obsługa użytkowników												
	Dodanie obsługi daty początkowej laboratoriów												
	Możliwość nadawania nazw maszynom												
Szablony laboratoriów	Basic oraz Kathara												
	R labs												
	P4												
Dokumentacja	Rozdział I oraz II												
	Rozdział III												
	Rozdział IV oraz V												

Rysunek 10: Harmonogram realizacji prac projektowych

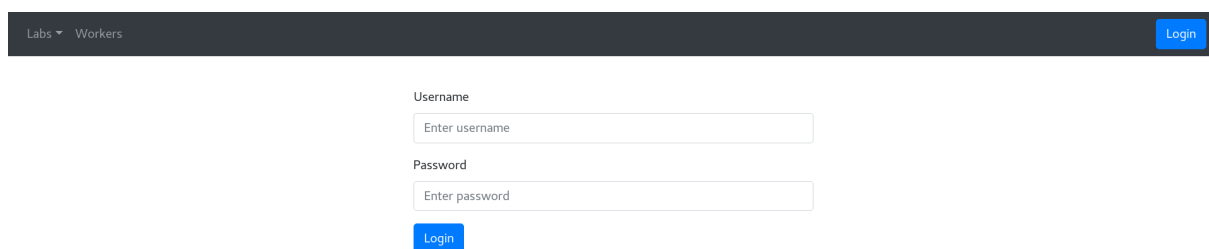
5. Wyniki projektu

Rozdział przedstawia rezultaty prac oraz prezentuje działanie podstawowych funkcjonalności systemu. Zawiera także propozycję dalszych zmian, które mogą zostać w przyszłości uwzględnione w systemie.

5.1. Produkt końcowy

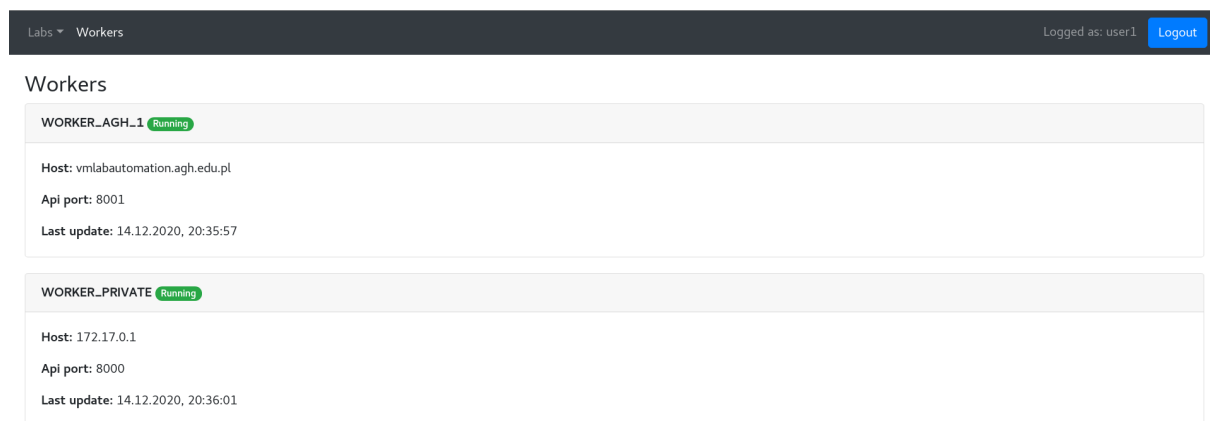
Finalne rozwiązanie jest gotowym systemem do automatycznego tworzenia odpowiednio przygotowanych środowisk do wykonania określonych zadań. Udostępnia ono przyjazny interfejs za pomocą, którego użytkownik może zrealizować przyjęte założenia. Poniżej zostały przedstawione podstawowe czynności, które użytkownik może wykonać korzystając z aplikacji.

W systemie istnieje mechanizm obsługi użytkowników. Przedstawiony na rysunku 11 zrzut ekranu prezentuje widok logowania do aplikacji.



Rysunek 11: Ekran logowania

Zarówno administrator aplikacji, jak i wszyscy użytkownicy posiadający dostęp, mogą wyświetlić listę dostępnych hostów, na których docelowo uruchamiane są laboratoria. Widok, zawarty na rysunku 12, przedstawia podstawowe informacje dotyczące *workerów*, a w szczególności ich status, który jest systematycznie aktualizowany.



Rysunek 12: Dostępne hosty (workery)

Każdy zalogowany użytkownik ma możliwość utworzenia laboratorium. Jedyną czynnością, którą musi wykonać jest wypełnienie formularza znajdującego się na zdjęciu poniżej, a dalsze kroki system wykona automatycznie. Formularz został umieszczony na rysunku 13.

The screenshot shows a web application interface with a dark header bar. On the left, there's a navigation menu with 'Labs' and 'Workers'. On the right, it says 'Logged as: user1' with a 'Logout' button. The main content area is titled 'Create a new lab' and contains a form with several fields and buttons.

Lab name: Lab kathara I rok, 16.11.2020

Lab type: Kathara

Start date: 2020-12-16 8:00 AM

Expiration date: 2020-12-20 8:00 PM

Description: Enter description

Virtual Machines: kowalskip, nowakj

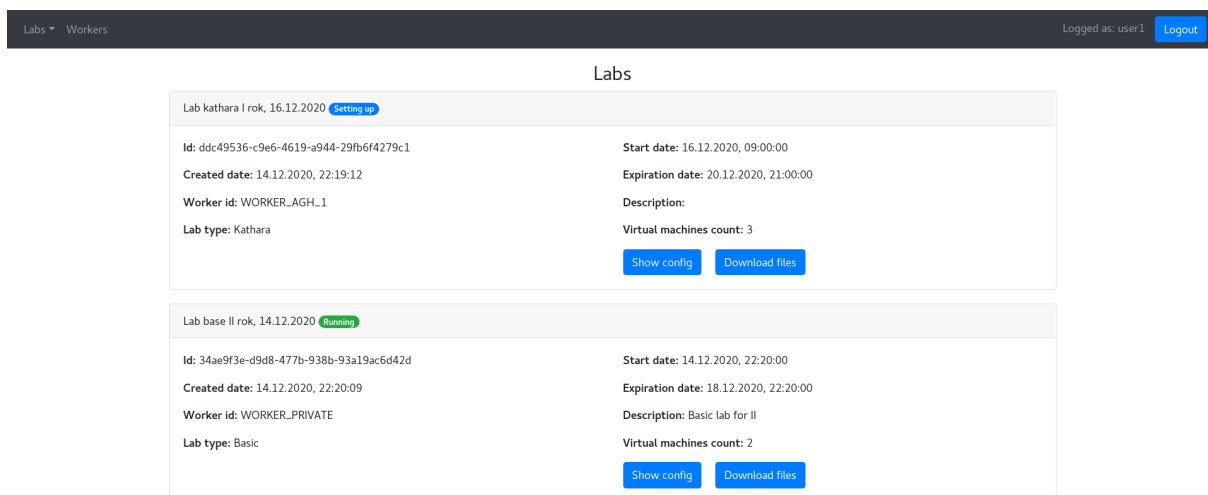
Enter machine name

Add next machine

Create

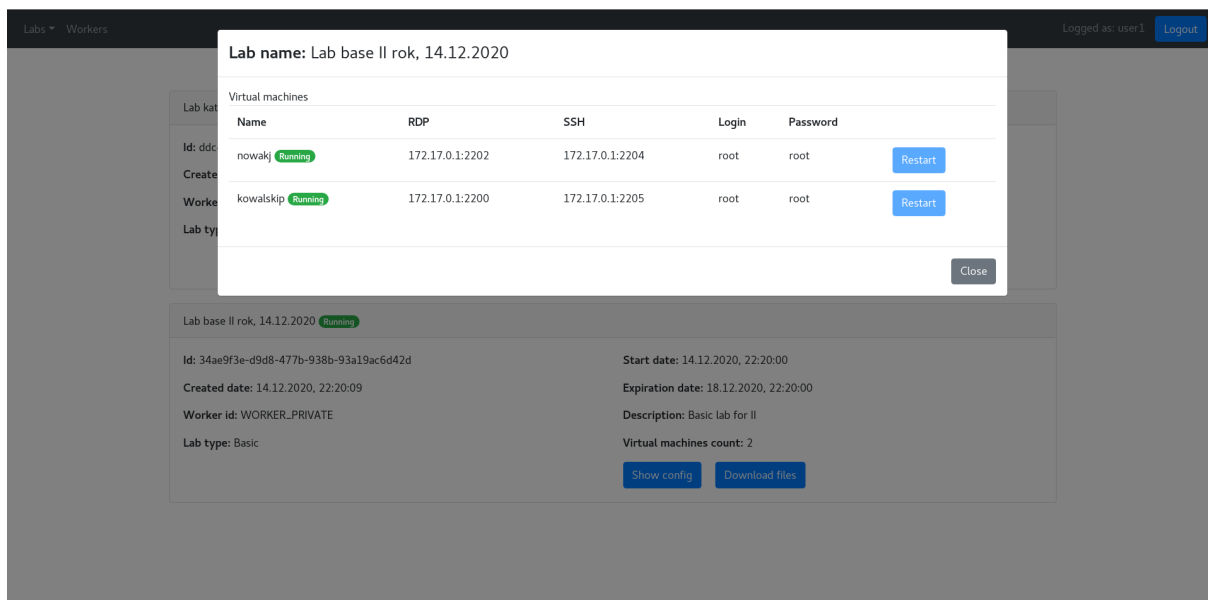
Rysunek 13: Ekran tworzenia laboratorium

Aplikacja dostarcza także funkcjonalność wyświetlania wszystkich laboratoriów, które stworzył dany użytkownik, wraz z podstawowymi informacjami na ich temat. Widok został zawarty na rysunku 14.



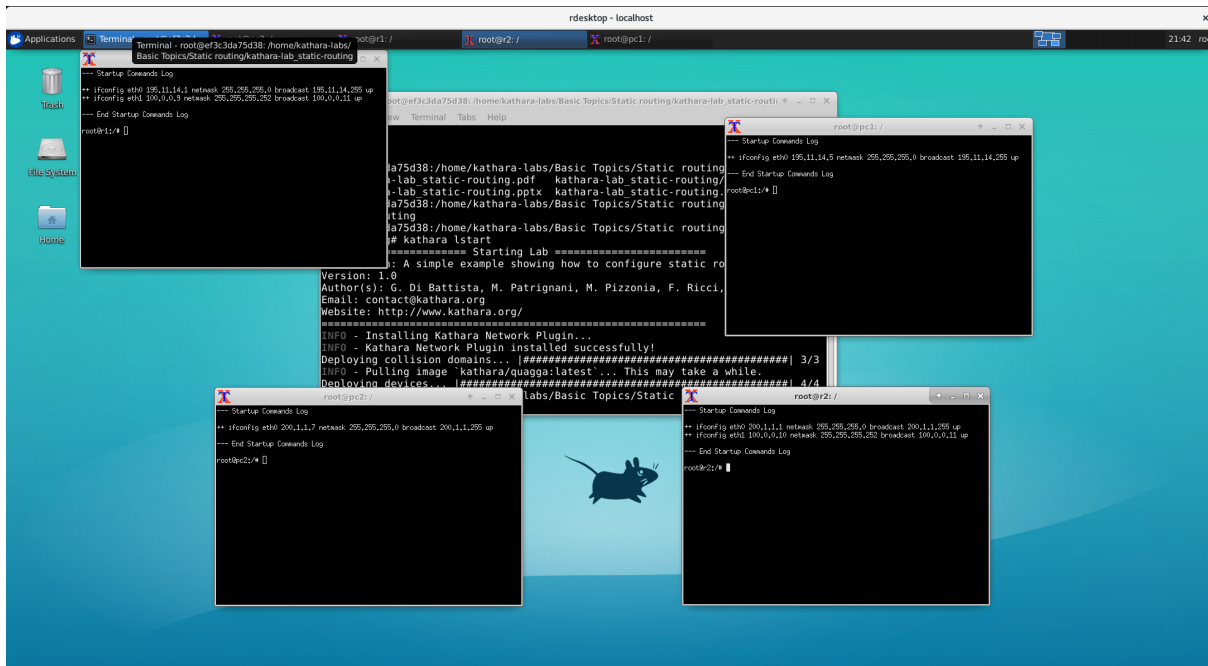
Rysunek 14: Ekran listy laboratoriów

Dla każdego laboratorium użytkownik ma możliwość sprawdzenia szczegółów dotyczących jego instancji, za pomocą okna dialogowego przedstawionego na rysunku 15. Znajdzie tam niezbędne informacje, które pozwolą na zalogowanie się do środowisk. Dla każdej maszyny dodany jest przycisk *Restart*, więc w przypadku awarii dowolnej z nich istnieje możliwość podjęcia próby automatycznej naprawy.



Rysunek 15: Ekran szczegółów pojedynczego laboratorium

W zadanym okresie czasu określonym przez datę początkową oraz końcową podczas tworzenia laboratorium, hosty udostępniają maszyny, na które można się zalogować. Zawierają one przygotowane środowisko niezbędne do wykonania wyznaczonego zadania. Poniżej zostało przedstawione przykładowe laboratorium dotyczące Kathary. Rysunek 16 prezentuje przykładowy ekran dostępny po zalogowaniu na maszynę.



Rysunek 16: Zdalny pulpit dla laboratorium

5.2. Realizacja wymagań funkcjonalnych

Logowanie do systemu

W aplikacji istnieje obsługa użytkowników. W obecnej wersji administrator jest odpowiedzialny za zarządzanie bazą osób korzystających z systemu. Do zalogowania niezbędny jest poprawny login oraz hasło. Logowanie wiąże się także z listą wyświetlanych laboratoriów, ponieważ podczas tworzenia nowego jest ono przypisywane do konkretnego użytkownika. Ekran logowania został przedstawiony na rysunku 11.

Automatyczne tworzenie laboratorium

Do utworzenia nowego laboratorium wymagane jest wyłącznie uzupełnienie prostego formularza, w celu przekazania niezbędnych danych do hosta (formularz został przedstawiony na rysunku 13). Można zatem stwierdzić, że cały proces jest w pełni zautomatyzowany.

Uruchomienie wielu instancji

System wspiera uruchamianie wielu maszyn dla pojedynczego laboratorium. Podczas tworzenia istnieje możliwość dodania kolejnych instancji oraz nadania im wybranych przez użytkownika nazw, co jest widoczne na rysunku 13. Dodatkowo podczas wyświetlania szczegółów danego laboratorium, osoba korzystająca z aplikacji otrzymuje także szczegółowe dane dotyczące poszczególnych maszyn, co zostało zaprezentowane na rysunku 15.

Udostępnienie przygotowanych szablonów do laboratorium

Po wybraniu danego rodzaju laboratorium podczas tworzenia, odpowiedni szablon zostanie automatycznie umieszczony na maszynie. Przykładowe laboratorium z Kathary zostało zaprezentowane na rysunku 16.

Połączenie z maszyną

Po utworzeniu laboratorium, zostaje ono automatycznie uwzględnione na liście wszystkich dostępnych dla zalogowanego użytkownika. Osoba korzystająca z aplikacji ma możliwość sprawdzenia na jakim adresie znajduje się maszyna (uwzględniając także porty *rdp* oraz *ssh*), a także danych dostępowych, takie jak login oraz hasło. Wykorzystując te dane można nawiązać połączenie z maszyną.

Zablokowanie dostępu do laboratorium po wyznaczonej dacie

Podczas tworzenia laboratorium wymagane jest podanie daty początkowej oraz końcowej (co zostało przedstawione na rysunku 13). Zatem nie tylko istnieje górne ograniczenie dostępu do maszyny, a także początkowe. Pozwala to na kontrolowanie czasu, w którym uczniowie będą mieli możliwość połączenia się z maszyną.

Pobranie plików

Osoba zalogowana do systemu ma możliwość pobrania plików ze stworzonego wcześniej laboratorium - na rysunku 14 przedstawiającym widok listy wszystkich laboratoriów widoczny jest przycisk *Download files*. Pliki dotyczące wybranego laboratorium są pobierane w postaci archiwum *zip* ze wszystkich instancji. Nauczyciel ma możliwość pobrania plików, nawet jeśli dostęp do maszyn został zablokowany ze względu na ustawioną datę końcową.

Restartowanie poszczególnych instancji w przypadku awarii

Na rysunku 15 przedstawiającym szczegóły wybranego laboratorium widoczny jest przycisk *Restart*. W przypadku działających maszyn jest on zablokowany, natomiast w przypadku niepowodzenia podczas uruchamiania laboratorium zostanie wyświetlony odpowiedni status danej maszyny oraz odblokowany, wspomniany wcześniej przycisk. Kliknięcie na niego spowoduje próbę ponownego uruchomienia maszyny. Możliwość resetowania jest dostępna tylko w przypadku, gdy warunki dotyczące daty początkowej oraz końcowej są spełnione.

Wyświetlenie listy dostępnych hostów

Każda osoba, która zaloguje się do aplikacji ma możliwość wyświetlenia listy dostępnych hostów. Oprócz podstawowej informacji, jaką jest nazwa workera, system udostępnia także takie informacje, jak adres hosta, port REST API oraz datę ostatniej aktualizacji stanu.

5.3. Realizacja wymagań нефункциональных

Prosty, intuicyjny interfejs użytkownika

Głównym założeniem przy tworzeniu interfejsu była jego intuicyjność i łatwość użytkowania. Na zdjęciu 13 został przedstawiony widok tworzenia laboratorium. Jak widać zawiera on niezbędne pola do wypełniania podczas tworzenia laboratorium. Na samej górze jest także widoczny pasek nawigacyjny, którego zadaniem jest ułatwienie użytkownikowi korzystania z przygotowanej aplikacji.

W celu zapewnienia dodatkowej wygody podczas korzystania z aplikacji zadbane także o detale, takie jak na przykład *date picker*, który ma ułatwić ustawienie odpowiedniej daty

początkowej oraz końcowej podczas tworzenia laboratoriów. Dodano także wskaźnik ładowania strony oraz walidację formularza. Zostało to przedstawione na rysunku 17.

Start date

2020-12-16 11:45 PM

December 2020						
Su	Mo	Tu	We	Th	Fr	Sa
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2

Time 23:45

(a) wybór daty



Create a new lab

Lab name

Enter lab name

Please enter a lab name

Lab type

Basic

Start date

2020-12-16 11:32 PM

Expiration date

2020-12-22 11:32 PM

Description

Enter description

Virtual Machines

Enter machine name

Add next machine

(c) walidacja

Rysunek 17: Interfejs użytkownika

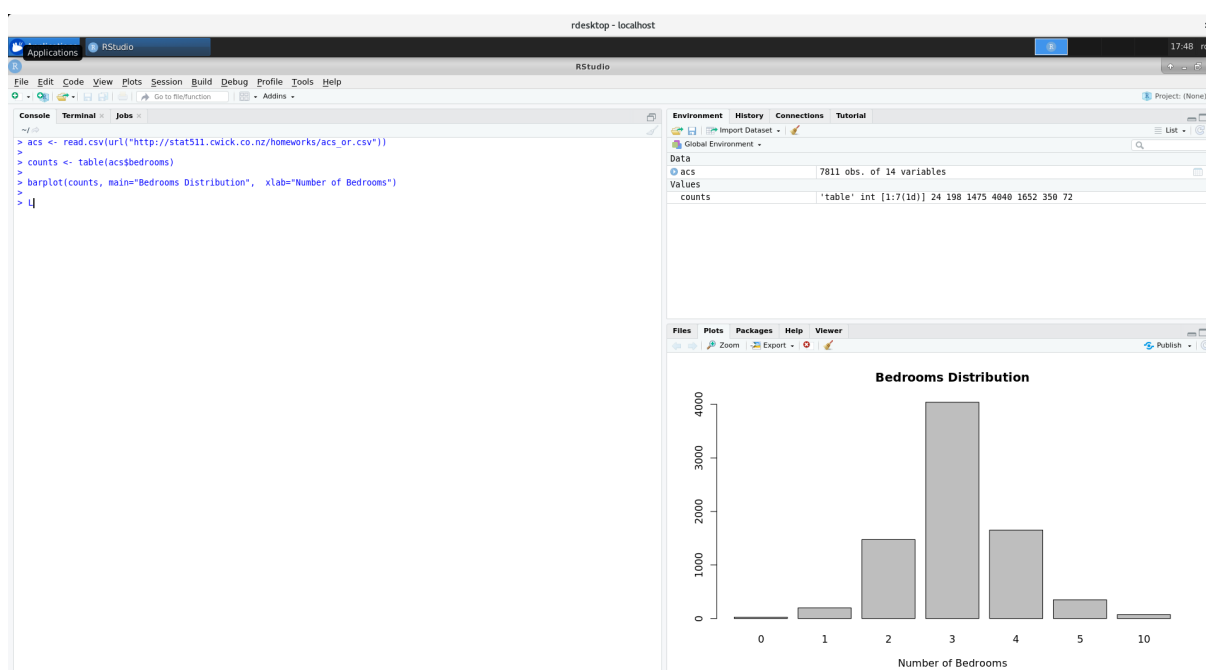
Zachowanie określonych praw dostępu

- Aby skorzystać z aplikacji, użytkownik musi podać odpowiedni login oraz hasło. W aktualnej wersji systemu administrator jest odpowiedzialny za zarządzanie bazą użytkowników. Powinno to zatem zapewnić, że z tego rozwiązanie będą korzystać tylko wyznaczone osoby.
- Nawiązanie połączenia z laboratorium wymaga posiadania odpowiedniego adresu oraz portu (*rdp* lub *ssh*). Te informacje są dostępne jedynie po zalogowaniu się do aplikacji. Dodatkowo konieczne jest podanie loginu oraz hasła do maszyny.
- Dodatkowym założeniem było udostępnienie maszyn na określony czas. Wprowadzono zatem konieczność ustawienia daty początkowej oraz końcowej laboratorium. Jednym z zadań workera jest cykliczne sprawdzanie warunków związanych z tymi datami i w określonych przypadkach albo uruchamianie, albo wstrzymywanie maszyn.

Otwartość na rozwój oraz inne technologie

Istnieje możliwość dalszego rozwoju stworzonego rozwiązania. Podział systemu na mniejsze komponenty z określonymi funkcjonalnościami powinien ułatwić dokładanie kolejnych. Propozycje zmian zostały opisane w punkcie 5.5.

Aplikacja zapewnia także otwartość na użycie innych technologii dzięki wykorzystaniu narzędzia Docker, które cieszy się dużą popularnością i dostarcza wiele możliwości. Ze względu na część niezbędnego oprogramowania, niezależnego od rodzaju laboratorium, został utworzony obraz dockerowy z podstawową konfiguracją. Można zatem w prosty sposób dodać nowy typ laboratorium z zupełnie inną technologią (niekoniecznie związana z sieciami programowalnymi) poprzez modyfikację, wcześniej wspomnianego, bazowego obrazu. Na rysunku 18 zostało przedstawione przykładowe laboratorium wykorzystujące język programowania R.



Rysunek 18: Przykład laboratorium wykorzystującego język programowania R

Skalowalność systemu

Odpowiednie zaplanowanie architektury systemu z uwzględnieniem podziału na moduły posiadające odrębne odpowiedzialności zapewnia, że aplikacja jest łatwo skalowalna. Wystarczy uruchomić moduł *workera* na nowym hoście, co pozwoli na automatyczne tworzenie instancji kolejnych laboratoriów na nim, odciążając jednocześnie pozostałe hosty.

5.4. Analiza zagrożeń

Zbyt duża ilość instancji przypadająca na hosta

Aplikacja od początku była projektowana w sposób skalowalny. Workery bardzo łatwo pozwalają rozszerzyć architekturę o dodatkowe zasoby. Rozwiązanie nie jest jednak idealne, ponieważ każdy host jest traktowany równo, co w przypadku słabego ogniwa mogłoby prowadzić do przeciążenia któregoś z hostów. Dystrybucja pracy jest bardzo szerokim zagadnieniem i istnieje tutaj

wiele możliwości rozwoju. Workery powinny zostać rozszerzone o możliwość monitorowania aktualnego zużycia dostępnych zasobów, natomiast aplikacja zarządzająca o odpowiednie algorytmy wyboru workera.

Stopień minimalizacji zagrożenia: średni.

Brak wsparcia dla różnych managerów maszyn wirtualnych

Wiele maszyn wirtualnych ze środowiskami zostało już utworzonych, dobrze byłoby móc je wykorzystać zamiast przerabiać na konkretną technologię (np. kontenery). Dzięki wykorzystaniu programu Vagrant możliwe jest wsparcie wszystkich wspieranych tam technologii wirtualizacji. Program ten jest również aktywnie rozwijany i rozszerzany o nowe podejścia. Z tego powodu w najbliższym czasie nie należy obawiać się braku wsparcia dla popularnego rozwiązania.

Stopień minimalizacji zagrożenia: wysoki.

Atak zewnętrzny

Aplikacja oraz udostępniane laboratoria ze względu na bliską współpracę z systemem działającym na hoście mogą być podatne na ataki. Kluczowym punktem jest tutaj fakt używania zagnieżdżonej konteneryzacji, która jest uruchamiana w przypadku kontenerów w trybie uprzywilejowanym. Problem należałoby zminimalizować przez przejście na “root-less containers”. Bezpieczniejsze są laboratoria przygotowane w oparciu o maszyny wirtualne.

Dodatkowo program worker komunikuje się bezpośrednio z systemem przez co jest narażony na ataki typu “injection”. Aktualny stan testów nie jest wystarczający, aby mieć pewność że jest w pełni bezpieczny.

Stopień minimalizacji zagrożenia: niski.

Brak stabilności technologii sieci programowalnych

W rozdziale 3.7 przedstawiony został przykład przygotowania laboratorium sieci programowalnych. Pokazał on, że nie jest to w pełni stabilna technologia, a także że dostępne narzędzia nie zawsze są w najnowszej wersji. Jednak system został zaprojektowany i zrealizowany, tak aby wspierać dowolną technologię, co pozwala też wprowadzać nowe rozwiązania z zakresu sieci programowalnych. Zostały też przygotowane konkretne szablony prezentujące tę możliwość.

Stopień minimalizacji zagrożenia: wysoki.

5.5. Rozwój systemu

System zgodnie z założeniami jest otwarty na modyfikacje, a możliwości dalszego rozwoju gotowego rozwiązania są ogromne. Poniżej przedstawiono tylko kilka z nich, które jednak zdecydowanie poprawiłyby funkcjonalność aplikacji:

Moduł do śledzenia zachowań

Administrator ma kontrolę nad tym, gdzie uruchamiane są laboratoria i nad zawartym w nich środowiskiem. Można by więc zawrzeć tam programy monitorujące czynności podejmowane przez użytkownika i zbierające statystyki. Służyłyby one do obserwacji działań osób korzystających z laboratoriów. Informacje, które mogłyby zostać przekazane do nauczyciela to na przy-

kład odwiedzone strony www lub czas spędzony nad poszczególnymi zadaniami. Zebrane dane mogłyby posłużyć do wyciągnięcia pewnych wniosków dotyczących przygotowania laboratorium.

Baza dostępnych laboratoriów

Nie każdy prowadzący posiada też umiejętności z zakresu przygotowywania opis maszyn, które mogłyby posłużyć za szablony. System wspiera uruchomienie dowolnego laboratorium, więc innym pomysłem rozwoju jest zbudowanie dużej bazy ogólnodostępnych laboratoriów. Umożliwiłoby to współdzielenie i ponowne wykorzystanie wielu szablonów oraz zaoszczędziło czas przygotowujących.

Przesłanie pliku przez aplikację

Kolejnym pomysłem na rozszerzenie systemu jest możliwość dodania pliku podczas wprowadzania danych dla nowego środowiska. Podczas tworzenia laboratorium plik zostałby przesłany i byłby do niego dostęp do zalogowania się do maszyny. W pliku mogłoby zostać przygotowane zadanie do rozwiązania.

Automatyczna weryfikacja rozwiązań

Aktualnie system dostarcza funkcjonalności pobierania plików z wybranego laboratorium, co powoduje, że nauczyciel musi ręcznie sprawdzić zadania wykonane przez uczniów. Moduł zbierania danych może zatem zostać rozwinięty o automatyczną weryfikację rozwiązania. Uprości oraz przyspieszy to proces oceniania wykonanych prac, co jednocześnie będzie miało przełożenie na szybszą wiadomość zwrotną dla osoby wykonującej ćwiczenie.

Przykładem takiej funkcjonalności są testy zawarte w frameworku Kathara. Dla innych środowisk mogą to być na przykład testy jednostkowe weryfikujące poprawność rozwiązania.

Statystyki maszyn

Głównym założeniem byłoby dostarczanie danych związanych z użyciem zasobów hostów. Umożliwiłoby to kontrolę nad dystrybucją zadań pomiędzy hostów oraz badanie wpływu wykonywanych akcji na ich aktualne wykorzystanie. Zminimalizowałoby przede wszystkim wspomniane zagrożenie przeciążenia hostów.

Maszyna jako plik

Istnieje wiele gotowych i wyeksportowanych maszyn, których środowisko nie jest jednak opisane za pomocą Vagrantfile lub Dockerfile. Można byłoby dodać wsparcie dla uruchamiania i powielania takich maszyn, aby wykorzystać dużą bazę, która już istnieje.

Innym pomysłem odnoszącym się również do tego punktu jest możliwość eksportu maszyny do pliku. Zamiast uruchomienia laboratoriów, mogą one zostać wyeksportowane do postaci maszyn wirtualnych i udostępnione bardziej zaawansowanym użytkownikom w postaci pliku, tak aby każdy z nich mógł uruchomić je niezależnie na swoim komputerze, nie obciążając zasobów serwera.

Panel administratora

Dodanie w systemie panelu zarządzania zdecydowanie mogłoby przyspieszyć i ułatwić pracę administratora. Przykładowe funkcjonalności, które mogłby dostarczać to między innymi zarządzanie użytkownikami oraz dostępnymi szablonami laboratoriów.

5.6. Podsumowanie

Celem naszej pracy inżynierskiej było przygotowanie systemu, który w prosty sposób umożliwia stworzenie odpowiednio przygotowanych środowisk z sieciami programowalnymi. Przy projektowaniu tego rozwiązania staraliśmy się, aby było ono uniwersalne i przydatne niezależnie od technologii, której miałyby dotyczyć laboratoria, co niewątpliwie zakończyło się powodzeniem.

Utworzona aplikacja dostarcza wszystkie funkcjonalności założone na początku pracy i jednocześnie ma wiele możliwości rozwoju. Dużym atutem dostarczonego rozwiązania jest jego architektura systemu, która została przygotowana w sposób, który powinien pozwolić na łatwe dodanie kolejnych funkcjonalności. Ze względu na swoją uniwersalność możemy uznać, że jest to dobry produkt bazowy, który można dalej rozwijać.

Gotowy produkt niezaprzeczalnie pomógłby rozwiązywać problemy, jakie zostały narzucone przez zdalny model nauczania. Dynamika tych zmian była bardzo duża i niewątpliwie potrzebne są nowe rozwiązania, które pomogły zarówno nauczycielom jak i uczniom. Przygotowana aplikacja mogłaby jednak pomóc również w modelu stacjonarnym, przyspieszając pracę i otwierając możliwości na poznanie nowych zagadnień.

W aktualnej postaci systemu występuje jednak wiele zagrożeń użytkowania oraz bezpieczeństwa, które zdecydowanie musiałyby zostać zminimalizowane przed udostępnieniem produktu szerszemu gronu odbiorców. Niemniej jednak stanowi on bardzo dobrą podstawę, a miejsca zagrożeń zostały dobrze zdefiniowane. W naszej opinii jest to rozwiązanie, które ma szansę sprawdzić się w rzeczywistych zastosowaniach.

A. Przykładowy szablon laboratorium

Poniżej zawarte zostały pliki służące jako szablon do laboratorium Kathara. Vagrantfile opisuje strukturę całego laboratorium - liczbę instancji, portu, foldery. Dockerfile natomiast opisuje jak wygląda pojedyncza instancja środowiska. Mogą one posłużyć jako wzór do tworzenia kolejnych laboratoriów.

```
vm_count = (ENV['VM_COUNT'] || '1').to_i

Vagrant.configure("2") do |config|

  config.vm.provider "docker" do |d|
    d.build_dir = "."
    d.has_ssh = true
    d.build_args = [ "--network", "host" ]
    d.create_args = [ "--privileged" ]
  end

  config.ssh.username="root"
  config.ssh.password="root"

  (1..vm_count).each do |i|
    config.vm.define "node-#{i}" do |node|
      node.vm.network "forwarded_port", guest: 3389,
                                         host: 3389,
                                         auto_correct: true
      node.vm.network "forwarded_port", guest: 22,
                                         host: 2222,
                                         auto_correct: true
      node.vm.synced_folder "./userdata/node-#{i}/",
                           "/home/lab", create: true
    end
  end
end
```

Listing 1: Vagrantfile

```
FROM lab-base

# Let's start with some basic stuff.
RUN apt-get update -qq && apt-get install -qqy \
    apt-transport-https \
    ca-certificates \
    curl \
    lxc \
    iptables

# Install Docker from Docker Inc. repositories.
RUN curl -sSL https://get.docker.com/ | sh

ENV DEBIAN_FRONTEND noninteractive
WORKDIR /scripts
COPY *.sh ./
RUN chmod +x *.sh
RUN ./bootstrap_kathara.sh

# Install the magic wrapper.
ADD ./wrapdocker /usr/local/bin/wrapdocker
RUN chmod +x /usr/local/bin/wrapdocker

# Kathara files
RUN apt-get update && apt-get install -y git zip gedit
RUN git clone https://github.com/KatharaFramework/Kathara-Labs \
    /home/kathara-labs

VOLUME /var/lib/docker
ADD etc /etc
```

Listing 2: Dockerfile

Spis rysunków

1	Wysokopoziomowa koncepcja systemu	5
2	Diagram sekwencji	8
3	Diagram przypadków użycia	10
4	Diagram architektury systemu	12
5	Struktura folderu przechowująca laboratoria	14
6	Diagram bazy danych	16
7	Diagram procesów CI/CD	19
8	Piramida testów	21
9	Diagram procesu wytwarzania kodu	24
10	Harmonogram realizacji prac projektowych	26
11	Ekran logowania	27
12	Dostępne hosty (workery)	28
13	Ekran tworzenia laboratorium	28
14	Ekran listy laboratoriów	29
15	Ekran szczegółów pojedynczego laboratorium	29
16	Zdalny pulpit dla laboratorium	30
17	Interfejs użytkownika	32
18	Przykład laboratorium wykorzystującego język programowania R	33

Spis tabel

1	Analiza zagrożeń	7
---	----------------------------	---

Materialy źródłowe

- [1] Automat-it.
<https://www.automat-it.com/environment-as-a-service>.
- [2] Azure.
<https://azure.microsoft.com/en-us/>.
- [3] Azure dev ops docs.
<https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops>.
- [4] G. Battista, G. Bonofiglio, V. Iovinella, and G. Lospoto. Kathará: A container-based framework for implementing network function virtualization and software defined networks. 2018.
<https://ieeexplore.ieee.org/document/8406267>.
- [5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, C. S. J. Rexford, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. 2014.
<http://www.sigcomm.org/node/3503>.
- [6] Cisco packet tracer.
<https://www.netacad.com/courses/packet-tracer>.
- [7] Cypress.
<https://www.cypress.io/>.
- [8] Docker.
<https://www.docker.com/>.
- [9] Docker compose docs.
<https://docs.docker.com/compose/>.
- [10] Dockerfile reference.
<https://docs.docker.com/engine/reference/builder/>.
- [11] Draw.io.
<https://app.diagrams.net/>.
- [12] Flask-restplus docs.
<https://flask-restplus.readthedocs.io/en/stable/>.
- [13] M. Fowler. The practical test pyramid. 2018.
<https://martinfowler.com/articles/practical-test-pyramid.html>.
- [14] Gitlab.
<https://about.gitlab.com/>.
- [15] Gns3.
<https://www.gns3.com/>.
- [16] Google meet.
<https://meet.google.com/>.
- [17] Haproxy.
<http://www.haproxy.org/>.

- [18] Jest.
<https://jestjs.io/>.
- [19] Kathara docs.
<https://www.kathara.org/>.
- [20] Konfiguracja maszyny wirtualnej środowiska p4.
<https://github.com/p4lang/tutorials/tree/master/vm>.
- [21] Małopolska chmura edukacyjna.
<https://e-chmura.malopolska.pl/>.
- [22] Microsoft teams.
<https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/education>.
- [23] Oci docs.
<https://opencontainers.org/>.
- [24] Overleaf.
<https://www.overleaf.com/>.
- [25] P4lang.
<https://github.com/p4lang>.
- [26] J. Petazzoni. Using docker-in-docker. 2015.
<https://github.com/jpetazzo/dind>.
- [27] Pytest.
<https://docs.pytest.org/en/stable/>.
- [28] Terraform.
<https://registry.terraform.io/>.
- [29] Trello.
<https://trello.com/>.
- [30] Vagrant docs.
<https://www.vagrantup.com/docs>.
- [31] Wipro.
<https://www.wipro.com/applications/environment-as-a-service/>.
- [32] Wiremock.
<http://wiremock.org/>.