

Politechnika Warszawska

W Y D Z I A Ł   M A T E M A T Y K I  
I   N A U K   I N F O R M A C Y J N Y C H



# Praca dyplomowa inżynierska

na kierunku Informatyka i Systemy Informacyjne

System do zdalnej pracy w środowisku graficznym wykorzystujący  
maszyny wirtualne QEMU z akceleracją sprzętową

Krzysztof Smogór

Numer albumu 298906

Piotr Widomski

Numer albumu 298919

promotor

dr inż. Marek Kozłowski

WARSZAWA 2022

.....

podpis promotora

.....

podpisy autorów

## **Streszczenie**

System do zdalnej pracy w środowisku graficznym wykorzystujący maszyny wirtualne QEMU z akceleracją sprzętową

Streszczam.

Lorem ipsum dolor sit amet, consetetur sadipscing elit, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

**Słowa kluczowe:** slowo1, slowo2, ...



## Abstract

Environment for remote work with Graphical User Interface using QEMU virtual machines with hardware acceleration

Konieczne jest załączenie wypełnionego oświadczenia o autorstwie pracy. By tego dokonać, skan (w formacie PDF) należy umieścić w folderze *scans* i nazwać go, np. `oswiadczenie_o_autorstwie_pracy.pdf` (w przypadku innej nazwy lub umieszczenia w innym folderze, konieczne jest adekwatne zmodyfikowanie ścieżki w komendzie je załączającej — patrz fragment kodu OŚWIADCZENIA).

**Keywords:** keyword1, keyword2, ...



Załącznik nr 1 do zarządzenia nr 109 /2021

Rektora PW z dnia 9 listopada 2021 r.

załącznik nr 5 do zarządzenia nr 42 /2020 Rektora PW



**Politechnika Warszawska**

.....  
miejscowość i data

.....  
imię i nazwisko studenta

.....  
numer albumu

.....  
kierunek studiów

## **OŚWIADCZENIE**

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

.....  
czytelny podpis studenta





Załącznik nr 3 do zarządzenia nr 109 /2021

Rektora PW z dnia 9 listopada 2021 r.

załącznik nr 9 do zarządzenia nr 42 /2020 Rektora PW



**Politechnika Warszawska**

.....  
miejsowość i data

.....  
imię i nazwisko studenta

.....  
numer albumu

.....  
Wydział i kierunek studiów

Oświadczenie studenta w przedmiocie udzielenia licencji  
Politechnice Warszawskiej

Oświadczam, że jako autor/współautor\* pracy dyplomowej pt. ....  
..... udzielam/nie udzielam\* Politechnice Warszawskiej  
nieodpłatnej licencji na niewyłączne, nieograniczone w czasie, umieszczenie pracy dyplomowej w elek-  
tronicznych bazach danych oraz udostępnianie pracy dyplomowej w zamkniętym systemie bibliotecznym  
Politechniki Warszawskiej osobom zainteresowanym.

Licencja na udostępnienie pracy dyplomowej nie obejmuje wyrażenia zgody na wykorzystywanie pracy  
dyplomowej na żadnym innym polu eksploatacji, w szczególności kopiowania pracy dyplomowej w całości  
lub w części, utrwalania w innej formie czy zwielokrotniania.

.....  
czytelny podpis studenta

\* niepotrzebne skreślić



## Spis treści

<b>1. Wstęp</b>	<b>11</b>
1.1. Opis problemu	11
1.2. Podobne rozwiązania	11
1.3. Wizja systemu	12
1.4. Istotne pojęcia	13
1.5. Wymaganie funkcjonalne	15
1.5.1. Nadzorca	15
1.5.2. Serwer wirtualizacji	17
1.5.3. Panel administratora	19
1.6. Wymaganie niefunkcjonalne	21
1.7. Analiza ryzyka	22
1.7.1. Omówienie zagrożeń	22
1.8. Podział pracy	24
<b>2. Opis rozwiązania</b>	<b>26</b>
2.1. Architektura systemu	26
2.1.1. Model systemu	27
2.1.2. Nadzorca	28
2.1.3. Serwer wirtualizacji	29
2.1.4. Aplikacja kliencka	29
2.1.5. Panel administratora	30
2.1.6. Broker wiadomości	30
2.2. Zewnętrzne narzędzia	31
2.2.1. Ansible	31
2.2.2. Vagrant	31
2.2.3. Libvirt z QEMU	31
2.3. Stany biznesowe	32
2.3.1. Maszyna wirtualna	32

2.3.2.	Serwer wirtualizacji . . . . .	33
2.3.3.	Użytkownik . . . . .	34
2.4.	Procesy biznesowe . . . . .	35
2.4.1.	Uzyskanie sesji . . . . .	35
2.4.2.	Kończenie sesji . . . . .	36
2.4.3.	Rozpoczęcie pracy serwera wirtualizacji . . . . .	36
2.5.	Komunikacja . . . . .	38
2.5.1.	Komunikacja wewnętrzna . . . . .	38
2.5.2.	Komunikacja zewnętrzna . . . . .	38
2.6.	Sekwencje komunikacji . . . . .	40
2.6.1.	Utworzenie sesji . . . . .	40
2.6.2.	Zakończenie sesji . . . . .	42
2.6.3.	Aktualizacja stanu . . . . .	42
2.6.4.	Włączenie maszyny . . . . .	42
2.6.5.	Wyłączenie maszyny . . . . .	43
2.7.	Wykorzystane technologie . . . . .	43
2.7.1.	Technologie realizacji strony klienckiej . . . . .	43
2.7.2.	Technologie realizacji strony serwerowej . . . . .	44
2.7.3.	Technologie testowania . . . . .	44
2.7.4.	Utrzymanie kodu . . . . .	45
2.8.	Wymagania systemu . . . . .	45
2.8.1.	Komunikacja sieciowa między modułami . . . . .	45
2.8.2.	Wymagania aplikacji klienckiej . . . . .	46
2.8.3.	Wymagania aplikacji nadzorcy i serwera panelu administracyjnego . . . . .	47
2.8.4.	Wymagania serwera wirtualizacji . . . . .	47
2.8.5.	Automatyzacja konfiguracji . . . . .	48
2.8.6.	Wymagania szablonu maszyny wirtualnej . . . . .	49
2.9.	Uruchomienie systemu . . . . .	49
2.9.1.	Pozyskanie aplikacji klienckiej . . . . .	49
2.9.2.	Zbudowanie kontenerów . . . . .	50
2.9.3.	Budowanie modułów . . . . .	50
2.9.4.	Konfiguracja aplikacji klienckiej . . . . .	51
2.9.5.	Konfiguracja panelu administracyjnego . . . . .	51
2.9.6.	Konfiguracja nadzorcy . . . . .	52

2.9.7.	Konfiguracja serwera wirtualizacji . . . . .	54
2.9.8.	Procedura uruchomienia . . . . .	59
2.9.9.	Parametry uruchomienia kontenera panelu administracyjnego . . . . .	59
2.9.10.	Parametry uruchomienia kontenera nadzorcy . . . . .	60
2.9.11.	Parametry uruchomienia kontenera serwera wirtualizacji . . . . .	61
2.9.12.	Przykład minimalnego systemu . . . . .	62
2.10.	Interakcja z systemem . . . . .	62
2.10.1.	Typowe działanie systemu . . . . .	62
2.10.2.	Funkcje panelu administracyjnego . . . . .	63
2.10.3.	Funkcje aplikacji klienckiej . . . . .	66
<b>3.</b>	<b>Analiza rozwiązania . . . . .</b>	<b>72</b>
3.1.	Testowana funkcjonalność . . . . .	72
3.1.1.	Brak komunikacji z nadzorcą . . . . .	72
3.1.2.	Utrata komunikacji z nadzorcą . . . . .	72
3.1.3.	Standardowe użycie systemu przez użytkownika . . . . .	73
3.1.4.	Standardowe użycie systemu przez użytkownika przy awarii nadzorcy . . . . .	73
3.1.5.	Podłączenie nowego serwera wirtualizacji . . . . .	74
3.1.6.	Podłączenie nowego nadzorcy . . . . .	74
3.1.7.	Odnutowanie utraty serwera wirtualizacji . . . . .	74
3.1.8.	Utrata komunikacji przy działającej sesji . . . . .	75
3.1.9.	Odzyskanie komunikacji przy działającej sesji . . . . .	75
3.2.	Środowisko testowe . . . . .	76
3.3.	Wykonane testy . . . . .	76
3.4.	Wyniki testów . . . . .	77
<b>4.</b>	<b>Podsumowanie . . . . .</b>	<b>78</b>
4.1.	Końcowy stan systemu . . . . .	78
4.2.	Możliwe ścieżki rozwoju . . . . .	79
4.2.1.	Panel administratora . . . . .	79
4.2.2.	Użycie konkretnych kart graficznych . . . . .	79
4.3.	Otwarte problemy . . . . .	79
4.3.1.	Konfiguracja połączenia RabbitMQ . . . . .	79
4.3.2.	Monitorowanie stanu połączenia klienta . . . . .	79
4.3.3.	Znane problemy wyścigów . . . . .	80



# 1. Wstęp

## 1.1. Opis problemu

Można aktualnie zaobserwować dużą zmianę w rynku pracy. Z powodu globalnej epidemii wiele firm zdecydowało się na zmianę pracy stacjonarnej na zdalną. Nawet po złagodzeniu obostrzeń, znaczna część miejsc pracy pozostała przy takim trybie, lub przyjęło hybrydową formę pracy. Taka forma pracy prowadzi jednak do pewnych utrudnień. Pracownicy mogą musieć łączyć się za pomocą funkcji zdalnego pulpitu z komputerami znajdującymi się w biurze. Może to wynikać z niewystarczającej wydajności sprzętu pracownika, lub dostępu do specyficznych programów lub zasobów. W takim wypadku komputer, z którym łączy się pracownik, musi być uruchomiony, a w przypadku awarii - zrestartowany. Dodatkowo taki dostęp może być wymagany przez ograniczony czas, co powoduje, że dużą część czasu spędza włączony, ale nieużywany.

Możliwym sposobem na złagodzenie tego problemu jest użycie zmniejszonej liczby komputerów, które mogą być używane przez większą liczbę pracowników jednocześnie, za pośrednictwem maszyn wirtualnych. Tym zmniejszamy liczbę działających maszyn, a zarządzanie może być rozwiązane za pomocą zdalnego operowania komputerem, na którym działają.

System stworzony w ramach tej pracy adresuje opisany problem. Rozwiązanie opiera się na tym wcześniej opisanym, jednocześnie rozbudowując je w sposób ułatwiający użytkowanie oraz zarządzanie.

## 1.2. Podobne rozwiązania

Jednym z bardziej znanych rozwiązań podobnych do naszego systemu jest system Citrix(<https://www.citrix.com/pl-pl/>). Oferuje on szeroki wachlarz usług dostępu do pulpitu zdalnych, aplikacji, usług w chmurze przeznaczony do pracy z dowolnego miejsca w sieci. Jedną z usług jest bardzo podobną do naszego systemu - Citrix Virtual Apps and Desktops[8]. Przy zakupie systemu można wybrać czy interesuje nas udostępnianie całego pulpitu zdalnego jako serwis (tzw. DaaS) albo po prostu zdalny dostęp do wybranego komputera.

System ten także posiada zbiór komputerów łączony w klaster oraz aplikacje balansującą, który równomiernie rozkłada zużycie wszystkich komputerów w klastrze. Możliwe jest w skorzystanie z gotowej infrastruktury zaoferowanej przez Citrixa albo wybudowanie własnego klastra.

Jedną z ważniejszych różnic Citrixa od naszego systemu jest większa wirtualizacja zasobów podłączonych do maszyny wirtualnej. W naszym systemie możliwe jest bardziej statyczne przyporządkowanie zasobów (np. karta graficzna), co pozwala na uzyskanie lepszej wydajności do nie typowych zastosowań biurowych (np. symulacje numeryczne w programach typu CAD). Dodatkowo nasz system pozwala łatwo wykorzystać istniejące komputery do stworzenia klastra. Konfiguracja każdego komputera w klastrze może być ustawiona oddzielnie, aplikacja nadzorcy odpowiednio zarządzi różnymi typami maszyn. Taki fakt może obniżyć koszty, jeżeli pracownicy często zmieniają swój tryb pracy.

### 1.3. Wizja systemu

Tworzony system ma za zadanie umożliwiać zdalną pracę za pomocą protokołu zdalnego pulpitu. System skierowany jest w stronę firm zatrudniających wielu pracowników, które chcą scentralizować sprzęt używany do pracy zdalnej.

Użytkownikami końcowym są pracownicy, którzy za pomocą okienkowej aplikacji klienckiej mogą uzyskać sesję do pracy zdalnej. Użytkownik podczas łączy się za pomocą protokołu zdalnego pulpitu z maszyną wirtualną uruchamiającą obraz systemu GNU/Linux. Uruchamianie i zarządzanie maszynami jest zadaniem aplikacji działającej na rzeczywistej maszynie, która udostępnia swoje zasoby maszynom wirtualnym. Aplikacja ta, oraz rzeczywista maszyna uruchamiająca ją, nazywana jest dalej serwerem wirtualizacji. Aplikacje te działają niezależnie od siebie i nie ma teoretycznego ograniczenia na ich liczbę w systemie. Komunikacją z użytkownikami oraz zarządzaniem systemem zajmuje się aplikacja nadzorcza. Ilość jej instancji również jest teoretycznie nieograniczona, co umożliwia balansowanie obciążeniem.

System pozwala na tworzenie maszyn wirtualnych różnych typów, czyli kombinacji zasobów systemowych udostępnianych dla maszyny wirtualnej, oraz faktu czy ma ona bezpośredni dostęp do karty graficznej maszyny, na której pracuje. Do używania systemu użytkownik musi posiadać konto w systemie katalogowym, który umożliwia użytkownikom dostęp do własnego folderu domowego na każdej maszynie. System katalogowy nie jest ujęty w obrębie systemu, ale jego poprawna konfiguracja jest wymagana do użytkowania systemu.

System udostępnia panel administracyjny w postaci strony WWW umożliwiający podgląd obciążenia i stanu systemu przez upoważnione osoby. Komunikacja aplikacji klienckiej z aplika-



#### 1.4. ISTOTNE POJĘCIA

cją nadzorczą oraz panel administratora wykorzystują komunikację za pomocą protokołu HTTP. Możliwe jest użycie szyfrowanego protokołu HTTPS, pod warunkiem użycia poprawnych certyfikatów SSL/TSL.

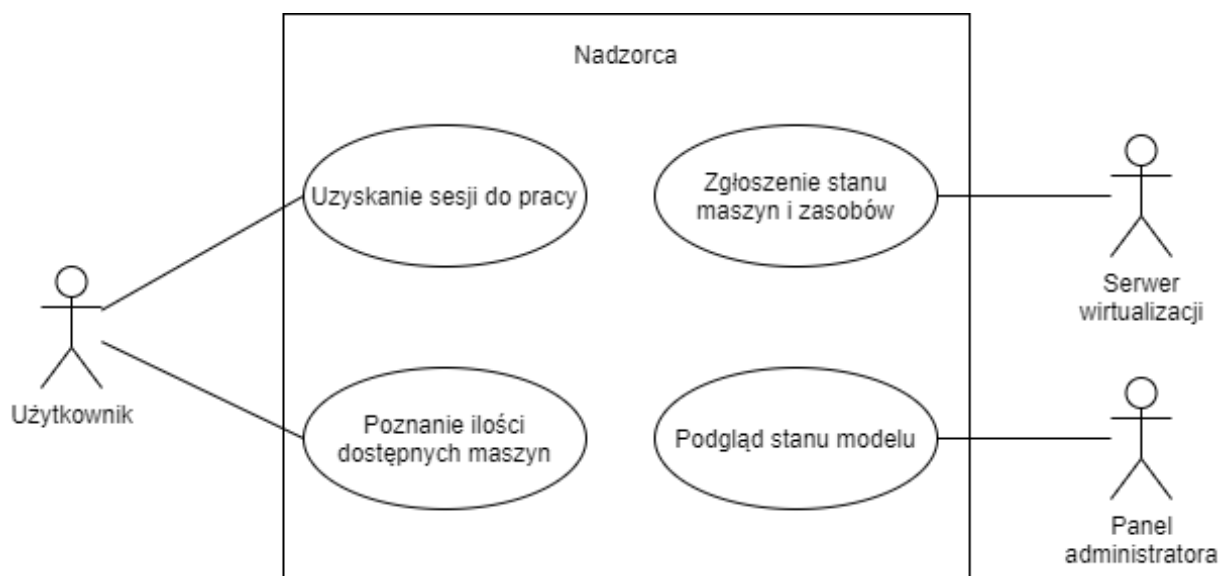
#### 1.4. Istotne pojęcia

- Aplikacja kliencka - aplikacja okienkowa uruchamiana na komputerze użytkownika, która umożliwi komunikację z systemem oraz uruchomienie zewnętrznego programu implementującego protokół RDP.
- Aplikacja nadzorcza (Nadzorca) - aplikacja, która przetwarza zapytania od aplikacji klienckiej oraz komunikuje się ze wszystkimi serwerami wirtualizacji. Na podstawie tych informacji buduje model zajętości każdego z serwerów wirtualizacji oraz decyduje kiedy, i na którym serwerze, trzeba uruchomić nowe maszyny wirtualne. Decyduje również, do której wirtualnej maszyny ma podłączyć się użytkownik proszący o utworzenie sesji.
- Serwer wirtualizacji - komputer, który udostępnia swoje zasoby (rdzenie procesora, karty graficzne, pamięć RAM oraz przestrzeń dyskową) w postaci uruchamianych na nim maszyn wirtualnych. Komputer ten uruchamia aplikację, która odpowiada na zapytania aplikacji nadzorczej oraz wykonuje operacje na maszynach wirtualnych (uruchamianie i wyłączanie). Komputer może uruchamiać co najwyżej jedną aplikację, dlatego zarówno komputer, jak i aplikację, nazywamy serwerem wirtualizacji.
- Maszyna wirtualna CPU - maszyna systemowa emulująca, lub para-emulująca, sprzęt i służąca do uruchamiania systemu operacyjnego. Udostępnia użytkownikowi podstawowe zasoby (procesor, pamięć RAM i przestrzeń dyskową). Uruchamiana jest na serwerze wirtualizacji z liczbą zasobów określoną w konfiguracji. Maszyna wirtualna uruchamia system operacyjny GNU/Linux (ArchLinux).
- Maszyna wirtualna GPU - maszyna analogiczna do maszyny wirtualnej CPU. Wyróżnia się przekazaną na wyłączność, za pośrednictwem mechanizmu GPU Passthrough, kartą graficzną podłączoną do serwera wirtualizacji.
- RDP - protokół zdalnego dostępu do pulpitu od firmy Microsoft[15]. Maszyny wirtualne uruchamiają serwer RDP(XRDP - <http://xrdp.org/>), który umożliwia zdalną pracę za pośrednictwem protokołu RDP.

- Sesja - jednorazowy dostęp użytkownika do systemu oraz maszyny wirtualnej. Utworzenie sesji wiąże się z przypisaniem do użytkownika konkretnej maszyny wirtualnej, na której będzie pracować. Sesja kończy się w przypadku, gdy użytkownik poinformuje system o zakończeniu pracy lub gdy minie czas oczekiwania na odzyskanie połączenia jego utracie.
- Vagrant-box[13] - przygotowany wcześniej obraz maszyny wirtualnej, który umożliwia zmianę dostępnych zasoby. Uruchamiają się bardzo powtarzalnie w środowisku programu Vagrant. Obrazy te używane są do tworzenia maszyn wirtualnych.
- Ansible playbook[10] - skrypt konfiguracyjny dla systemu operacyjnego, który umożliwia parametryzację oraz wykonywanie podczas uruchamiania Vagrant-boxa.
- Panel administratora - aplikacja przeglądarkowa, która umożliwia administratorowi systemu podgląd listy serwerów wirtualizacji znajdujących się w systemie oraz zajętości zasobów.
- Konto użytkownika - profil użytkownika w systemie, do którego ma dostęp na każdej maszynie wirtualnej. Używając przygotowanych wcześniej danych logowania może za ich pomocą logować się do maszyn wirtualnych. Przechowywane są w zewnętrznym (poza opisanym systemem) systemie katalogowym.
- Katalog użytkownika - prywatny folder dostępny dla użytkownika na każdej maszynie wirtualnej. Przechowywany na zewnętrznym (poza opisanym systemem) dysku sieciowym.
- Konfiguracja stała - konfiguracja maszyny wirtualnej, która nie zmienia się w zależności od miejsca uruchomienia. Docelowo ta konfiguracja ma być zapisana w Vagrant-boxie. W razie potrzeby można ją także zdefiniować w odpowiednim Ansible playbooku.
- Konfiguracja zmienna - konfiguracja maszyny wirtualnej, która zmienia się w zależności od miejsca uruchomienia. Jest definiowana w odpowiednim Ansible playbooku uruchamianym przy każdym włączeniu maszyny.

## 1.5. Wymaganie funkcjonalne

### 1.5.1. Nadzorca

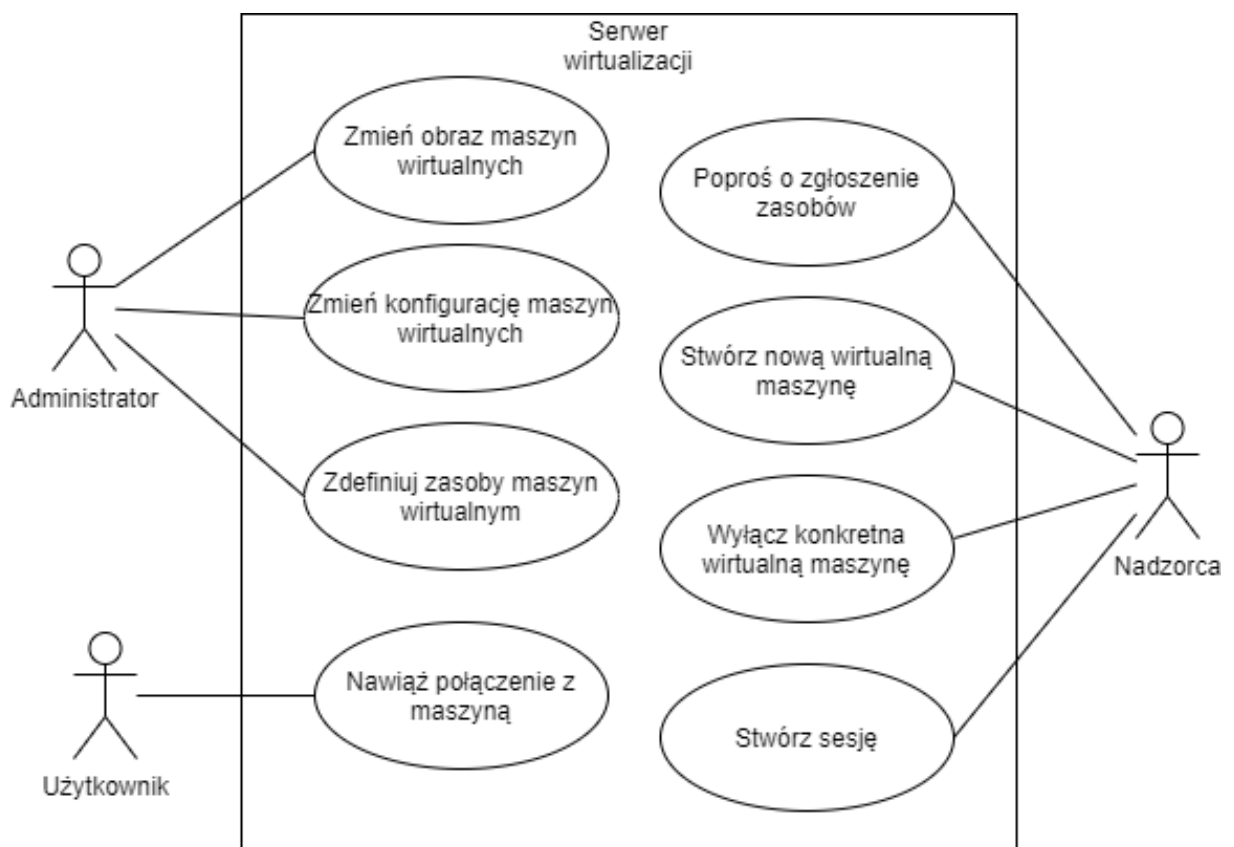


Rysunek 1.1: Przypadki użycia aplikacji nadzorczej

Tablica 1.1: Przypadki użycia aplikacji nadzorczej

Aktor	Nazwa	Opis	Odpowiedź systemu
Użytkownik	Uzyskanie sesji do pracy	Uzyskanie sesji do pracy na maszynie wirtualnej CPU lub GPU	Do użytkownika zostaje przydzielona maszyna wirtualna oraz zestawione połączenie RDP. W przypadku, gdy utracił on połączenie, to przydzielana jest do niego poprzednio używana maszyna, jeżeli jego sesja nie została jeszcze umorzona.
	Poznanie ilości dostępnych maszyn	Wyświetlanie szacowanej ilości dostępnych maszyn każdego typu	Użytkownikowi zostaje wyświetlona szacowana liczba dostępnych maszyn obliczona na podstawie informacji o dostępnych zasobach każdego z serwerów wirtualizacji
Serwer wirtualizacji	Zgłoszenie dostępnych zasobów	Serwer zgłasza nadzorcy dostępne zasoby	Nadzorca wykorzystuje zgłoszone zasoby do wyliczania szacowanej liczby dostępnych maszyn oraz do balansowania obciążenia serwerów wirtualizacji
Panel administratora	Podgląd stanu modelu	Nadzorca udostępnia panelowi administratora stan zasobów systemu.	Panel administratora wykorzystuje uzyskane dane do wygenerowania raportu o stanie systemu dla administratora wirtualizacji

1.5.2. Serwer wirtualizacji

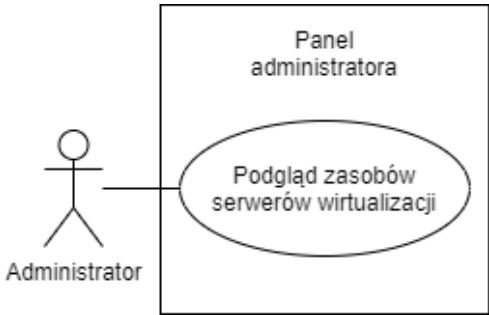


Rysunek 1.2: Przypadki użycia serwera wirtualizacji

Tablica 1.2: Przypadki użycia serwera wirtualizacji

Aktor	Nazwa	Opis	Odpowiedź systemu
Użytkownik	Nawiązanie połączenia z maszyną	Użytkownik nawiązuje połączenie z maszyną wirtualną	Maszyna wirtualna zostaje zajęta przez użytkownika; serwer wirtualizacji rozpoczyna monitorowanie, czy sesja wciąż trwa
Nadzorca	Poproś o zgłoszenie zasobów	Nadzorca wysyła do wszystkich serwerów wirtualizacji prośbę o zgłoszenie swoich używanych i wolnych zasobów	Serwer wirtualizacji informuje nadzorcę o stanie swoich zasobów
	Stwórz nową wirtualną maszynę	Nadzorca prosi serwer wirtualizacji o stworzenie nowej wirtualnej maszyny dla danego użytkownika na wybranym typie maszyny	Serwer wirtualizacji tworzy wirtualną maszynę i udostępnia możliwość połączenia się z nią
	Wyłącz konkretną wirtualną maszynę	Nadzorca prosi serwer wirtualizacji aby wyłączył konkretną wirtualną maszynę.	Serwer wirtualizacji wyłącza konkretną wirtualną maszynę oraz pilnuje aby na pewno się wyłączyła.
Administrator	Zmień obraz maszyn wirtualnych	Zmiana obrazu źródłowego maszyn wirtualnych	Zdefiniowany przez administratora vagrant-box jest używany przez serwery wirtualizacji
	Zmień konfigurację maszyn wirtualnych	Zmiana zmiennej konfiguracji maszyn wirtualnych	Zmodyfikowany ansible playbook jest używany przez serwery wirtualizacji
	Zdefiniuj zasoby maszyn wirtualnych	Zmiana ilości zasobów przydzielanych na każdy z typów maszyn wirtualnych oraz łączną ilość zasobów przeznaczonych na maszyny	Zmodyfikowana konfiguracja zasobów będzie wykorzystywana przez serwer wirtualizacji przy kolejnym uruchomieniu

1.5.3. Panel administratora



Rysunek 1.3: Przypadki użycia panelu administratora

Tablica 1.3: Przypadki użycia panelu administratora

Aktor	Nazwa	Opis	Odpowiedź systemu
Administrator	Podgląd zasobów serwerów wirtualizacji	Wyświetlanie wolnych oraz zajętych zasobów serwerów wirtualizacji	Wyświetlenie zasobów poszczególnych serwerów wirtualizacji, liczby zajętych maszyn oraz szacowanej liczby wolnych maszyn





## 1.6. Wymaganie niefunkcjonalne

Tablica 1.4: Wymagania niefunkcjonalne

Grupa wymagań	Nr wymagania	Opis
Użytkowanie (Usability)	1	Aplikacja kliencka ma działać na systemach operacyjnych MS Windows (Windows 10) oraz GNU/Linux (ArchLinux). Aplikacja na systemach GNU/Linux wymaga zainstalowanego klienta RDP zgodnego z XRDP <sup>1</sup> .
	2	Aplikacja kliencka musi udostępniać możliwość użycia własnego klienta RDP do nawiązania połączenia z maszyną wirtualną
	3	Maszyny wirtualne muszą mieć dostęp do systemu przechowującego konta użytkowników wraz z ich katalogami domowymi
Nieawodność (Reliability)	4	System musi być odporny na awarie poszczególnych serwerów wirtualizacji i kontynuować działanie w sposób niezauważalny dla użytkowników nie używających danego serwera.
	5	Awaria nadzorcy może spowodować uniemożliwienie rozpoczęcia nowych sesji, ale nie może przerwać istniejących sesji
Wydajność (Performance)	6	Łącznie zużywane zasoby przez maszyny wirtualne na poszczególnym serwerze wirtualizacji nie mogą przekroczyć wcześniej zdefiniowanych limitów
	7	Nadzorca musi balansować obciążenie serwerów wirtualizacji
	8	W systemie zawsze musi istnieć jedna działająca maszyna wirtualna nie połączona z żadną sesją, aby można było ją szybko przydzielić użytkownikowi
	9	Zwolnione maszyny wirtualne, które nie są wykorzystywane jako zapas, muszą być wyłączane
Utrzymanie (Supportability)	10	Możliwe jest działanie więcej niż jednego nadzorcy w systemie, w celu zwiększenia dostępności lub przeprowadzenia prac utrzymaniowych

## 1.7. Analiza ryzyka

Tablica 1.5: Analiza ryzyka

<p>Mocne strony</p> <ul style="list-style-type: none"> <li>• Łatwa skalowalność pod względem liczby sesji w systemie</li> <li>• Wiele rozwiązań Open Source</li> <li>• Elastyczność pod względem konfiguracji</li> <li>• Tańsze rozwiązanie niż kupno stacji roboczych</li> </ul>	<p>Słabości</p> <ul style="list-style-type: none"> <li>• System trudny w konfiguracji</li> <li>• Potrzeba wymiany sprzętu komputerowego</li> <li>• Krótki czas rozwoju systemu</li> <li>• Ograniczenie doświadczenie twórców systemu</li> <li>• Małe prawdopodobieństwo wsparcia projektu po zakończeniu prac</li> </ul>
<p>Okazje</p> <ul style="list-style-type: none"> <li>• Grupa docelowa to firmy z dużą ilością stacji roboczych</li> <li>• Zwiększenie zapotrzebowania na prace zdalną na rynku pracy</li> </ul>	<p>Zagrożenia</p> <ul style="list-style-type: none"> <li>• Istnienie konkurencji ugruntowanej na rynku</li> <li>• System w dużej mierze oparty o oprogramowanie rozwijane przez inne organizacje</li> </ul>

### 1.7.1. Omówienie zagrożeń

- System trudny w konfiguracji - wysoko prawdopodobne

Można temu zaradzić poprzez udostępnienie dokładnej dokumentacji lub ścisłą współpracę z klientem przy wdrażaniu systemu.

Wartość: duża

- Potrzeba wymiany sprzętu komputerowego - średnio prawdopodobne

Klient może potrzebować wymienić aktualne stacje robocze na terminale oraz zainwestować w sprzęt serwerowy. Jednak gdy klientami będą firmy, które mają dużo pracowników pracujących spoza biura, lub dopiero tych pracowników pozyskują, to kupno terminali i

## 1.7. ANALIZA RYZYKA

serwerów powinno być bardziej zachęcające niż kupno stacji roboczych.

Wartość: średnia.

- Krótki czas rozwoju systemu - wysoko prawdopodobne

Czas rozwoju systemu jest bardzo ograniczony. Aby pomimo tego ograniczenia działał on w sposób akceptowalny powinniśmy skupić się na dobrym przedyskutowaniu i opisanu kluczowych modułów systemu. W czasie projektu należy pilnować aby nie dodawać nadmiarowych funkcjonalności do systemu. W czasie implementacji krytyczne będzie dokładne zaplanowanie aplikacji pod kątem testowania automatycznego. Ułatwi to wyłapywanie prostych błędów jeszcze we wczesnej fazie projektu.

Wartość: wysoka

- Ograniczone doświadczenie twórców systemu - pewne

Jedynym sposobem na ograniczenie ryzyka jest rozważna implementacja.

Wartość: średnia

- Małe prawdopodobieństwo wsparcia projektu po zakończeniu prac - wysoko prawdopodobne

Trudno teraz przewidzieć co się stanie z projektem po zakończeniu prac. Jednak prawdopodobnie twórcy systemu zajmą się innymi projektami. Można jedynie dokładnie komentować kod i pokrywać jak najwięcej jego części testami. Wtedy inne osoby będą w stanie szukać błędów albo próbować w taki sposób uzupełnić brakującą wiedzę o systemie.

Wartość: niska

- Istnienie konkurencji ugruntowanej na rynku - bardzo prawdopodobne

Konkurencyjne systemy oferujące podobne rozwiązania są już dobrze ugruntowane na rynku i przetestowane. Nasz system może spróbować konkurować jedynie z nimi ceną implementacji oraz elastycznością.

Wartość: średnia

- System w dużej mierze oparty o oprogramowanie rozwijane przez inne organizacje - nisko prawdopodobne

W czasie życia systemu mogą pojawić się błędy w oprogramowaniu nie rozwijanym w ramach naszego systemu. naprawa takich błędów może trwać bardzo długo. Pewnym sposobem wsparcia takiego systemu jest własnoręczne poprawianie błędów w zewnętrznym oprogramowaniu i zgłaszanie ich do odpowiedniej organizacji. Do czasu zastosowania poprawki jest możliwość korzystania z wersji, na którą nanieśliśmy własną poprawkę.

Wartość: wysoka

## 1.8. Podział pracy

*Jest to wstępna wersja podziału pracy!*

W czasie realizacji systemu podzieliśmy się pracą:

- Krzysztof Smogór - aplikacja serwera wirtualizacji (w tym integracja z Vagrantem oraz libvirtem) oraz aplikacja nadzorcy (przetwarzanie informacji o systemie oraz udostępnienie jej dla aplikacji klienckich)

## 1.8. PODZIAŁ PRACY

- Piotr Widomski - aplikacja kliencka (wraz z integracją z klientami RDP), aplikacja panelu administracyjnego oraz komunikacja pomiędzy modułami.

W czasie pisania pracy dyplomowej podział był następujący:

- Krzysztof Smogór - Analiza rozwiązania (3) oraz część opisu rozwiązania związana z konfiguracją i wymaganiami (2.7 - 2.10).
- Piotr Widomski - Wstęp (1), Podsumowanie (4) oraz teoretyczna część opisu rozwiązania (2.1 - 2.6)

## 2. Opis rozwiązania

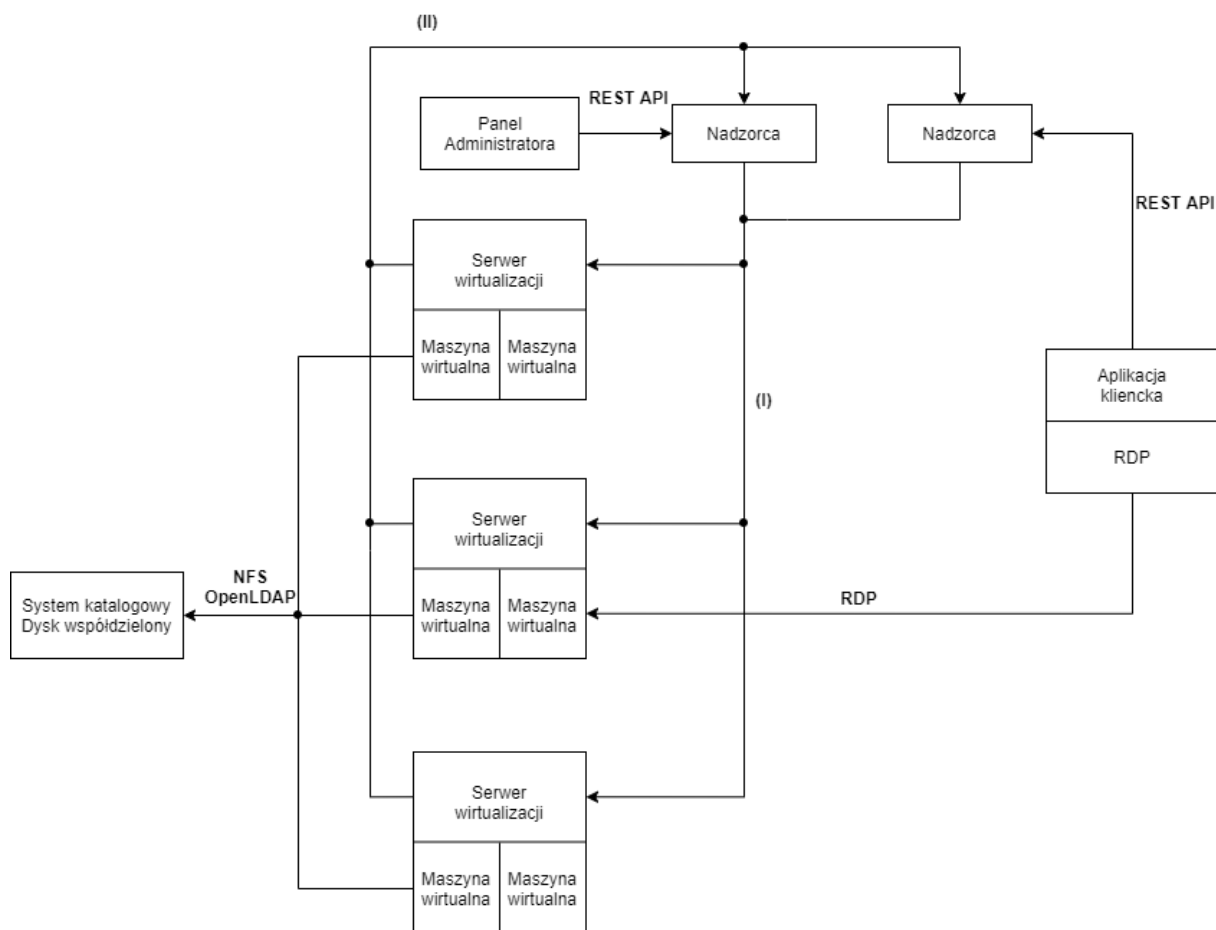
### 2.1. Architektura systemu

Opracowywany system składa się z następujących modułów:

- nadzorcy,
- serwera wirtualizacji,
- aplikacji klienckiej,
- panelu administratora,
- brokera wiadomości,
- systemu katalogowego,
- dysku współdzielonego.

Schematyczny obraz systemu przedstawia poniższy rysunek.

## 2.1. ARCHITEKTURA SYSTEMU



Rysunek 2.1: Schematyczna architektura systemu

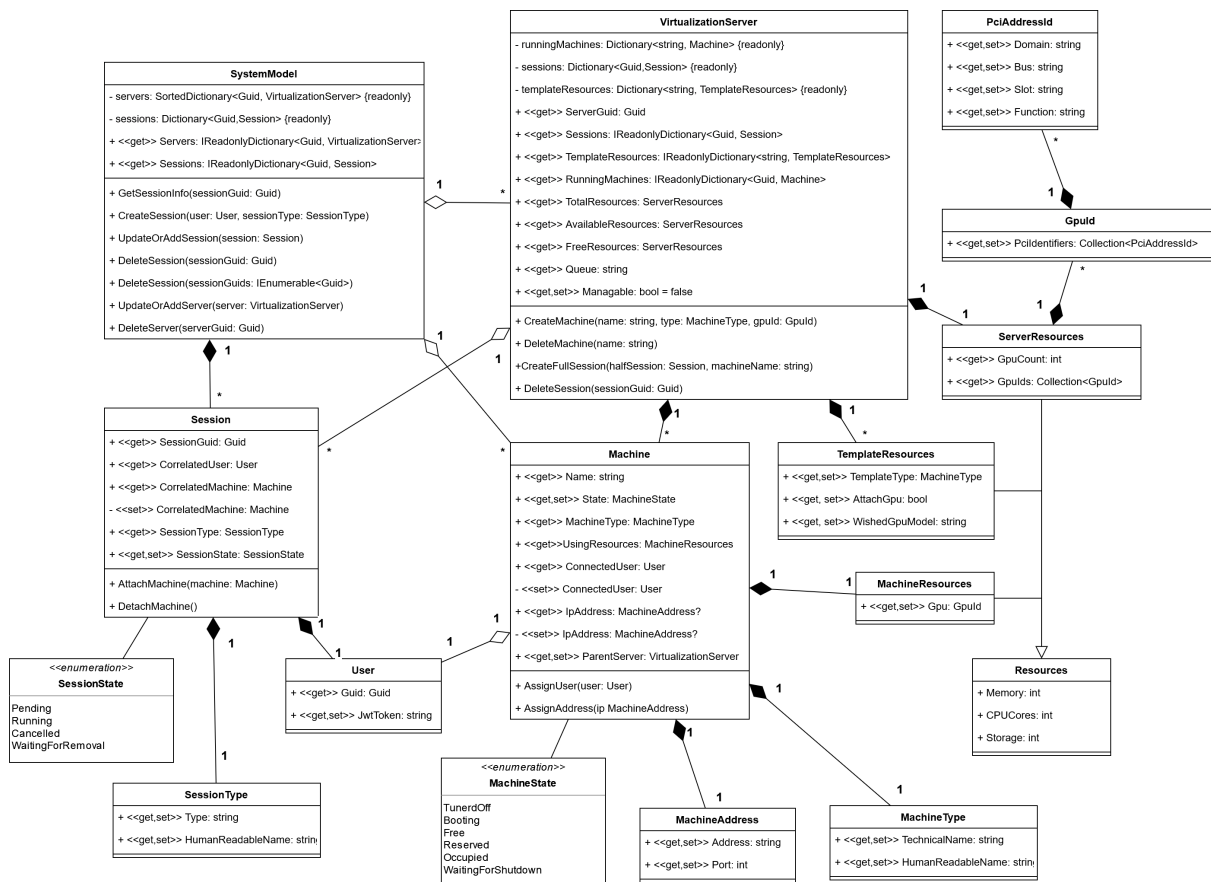
Połączenia oznaczone liczbami rzymskimi oznaczają kolejki komunikacji za pośrednictwem brokera wiadomości, które opisane zostały w punkcie jemu poświęconym. Z założenia system powinien móc skalować się w dwóch wymiarach, to znaczy:

1. Zwiększanie liczby serwerów wirtualnych - zwiększenie liczby istniejących jednocześnie sesji.
2. Zwiększenie liczby nadzorców - zwiększenie liczby obsługiwanych klientów jednocześnie oraz niezawodności systemu.

### 2.1.1. Model systemu

W celu zarządzania systemem, każdy z nadzorców musi posiadać dokładną wiedzę o jego aktualnym stanie. Informacje te przechowuje w strukturze nazywanej dalej modelem systemu. Ważnym jest, że klasy modelu nie wykonują żadnych akcji poza modyfikacją przechowywanych danych. Wszelkie metody służą jedynie do zmiany stanu przechowywanego modelu w celu dopasowania go do rzeczywistego stanu, na podstawie otrzymanych danych. Przyczyną takiego stanu, jest pierwsze z założeń komunikacji opisane w 2.5.1. Sprawia ono, że zmiana modelu musi być

następstwem pewnej akcji wykonanej przez serwer wirtualizacji.



Rysunek 2.2: Schemat klas modelu systemu

Główną klasą modelu jest **SystemModel** zawierający informacje o aktualnie działających serwerach wirtualizacji oraz aktywnych sesjach. Klasa **VirtualizationServer** modeluje pojedynczy serwer wirtualizacji, jego zasoby, maszyny wirtualne oraz obsługiwane sesje. Przechowuje ona również informacje wymagane do komunikacji z daną instancją serwera.

Klasa **Resources** oraz jej pochodne opisują zasoby systemowe i są używane do przedstawienia zarówno zasobów maszyny jak i całego serwera wirtualizacji. Jej klasa pochodna **TemplateResources** służy do przechowywania informacji o zasobach potrzebnych do utworzenia maszyny danego typu.

### 2.1.2. Nadzorca

Aplikacja mająca za zadanie obsługiwać komunikację z aplikacjami klienckimi oraz wysyłać polecenia do serwerów wirtualizacji. Udostępnia REST API służące do komunikacji z aplikacjami klienckimi. Do komunikacji z serwerami wirtualizacji wykorzystuje kolejki.

Nadzorca przechowuje wewnętrznie model systemu zawierający informację o działających serwerach wirtualizacji i stanie ich maszyn. Na podstawie tego modelu moduł stwierdza, do której



## 2.1. ARCHITEKTURA SYSTEMU

maszyny przypisać nowo utworzoną sesję. Wewnętrzne procesy skupione są wokół zmian modelu. Jeżeli proces wysłał do serwera wirtualizacji prośbę o zmianę stanu, to dalsze procesowanie odbywa się, gdy stan modelu został zaktualizowany, i na podstawie jego stanu podejmowane są decyzje.

Dzięki zastosowaniu kolejek oraz zasad komunikacji w systemie może istnieć więcej niż jeden nadzorca. Instancje nadzorców działają niezależnie od siebie i przechowują identyczny model systemu. Dzięki temu uzyskujemy retencję i możemy zmniejszyć obciążenie poszczególnych nadzorców.

### 2.1.3. Serwer wirtualizacji

Zadaniem serwera wirtualizacji jest uruchamianie i zarządzanie maszynami wirtualnymi, z którymi łączy się użytkownik systemu. Komunikuje się on z nadzorcami i wykonuje operacje na maszynach wirtualnych zgodnie z żądaniami.

Moduł ten nie jest w stanie funkcjonować samodzielnie. Z tego powodu aplikacja nie uruchomi się, jeżeli nie jest w stanie nawiązać połączenia z aplikacją nadzorczą, a w przypadku ostatni nadzorca w systemie zakończy działanie, aplikacja również je zakończy, pod warunkiem że nie ma żadnych działających sesji.

Serwer wirtualizacji jest częścią systemu, która przechowuje realne zasoby udostępniane użytkownikom. System zaprojektowany jest w taki sposób aby teoretycznie nie było ograniczenia na liczbę serwerów wirtualizacji działających jednocześnie.

### 2.1.4. Aplikacja kliencka

Aplikacja okienkowa umożliwiającą użytkownikowi autoryzację, uzyskanie sesji oraz automatyczne rozpoczęcie połączenia. Komunikuje się z nadzorcą za pomocą REST API.

Proces uzyskania sesji z perspektywy aplikacji klienckiej zawiera:

1. Uzyskanie informacji o dostępnych typach i liczbie maszyn
2. Wybór typu maszyny
3. Oczekiwanie na utworzenie sesji
4. Nawiązanie połączenia RDP
5. Utrzymanie i monitorowanie stanu połączenia.

### 2.1.5. Panel administratora

Prosta aplikacja internetowa umożliwiająca administratorowi systemu podgląd stanu zużycia zasobów serwerów wirtualizacji.

### 2.1.6. Broker wiadomości

Komunikacje wewnątrz systemu, czyli pomiędzy serwerami wirtualizacji oraz nadzorcami, będzie realizowali poprzez kolejki wiadomości. W tym celu użyty został system RabbitMQ, który zajmuje się transportem wiadomości wewnątrz systemu oraz niezawodnością komunikacji między modułami.

Zdefiniowane zostały następujące kolejki wiadomości:

- (I) Kolejka kończąca się na każdym z serwerów wirtualizacji powielająca wiadomości między nich. Służy ona do wysyłania nie spersonalizowanych próśb od nadzorców do serwerów wirtualizacji.
- (II) Kolejka kończąca się na każdym z nadzorców powielająca wiadomości między nich. Służy ona do przesyłania informacji do nadzorców o zmianach wewnątrz serwera wirtualizacji.
- (III) Kolejka kończąca się wyłącznie na pojedynczym serwerze wirtualizacji. Liczba kolejek zgadza się z liczbą serwerów wirtualizacji aktywnych w systemie. Służą one do przesyłania spersonalizowanych wiadomości oraz sprawdzania, czy serwer wirtualizacji nadal pracuje po drugiej stronie. Skorzystamy z funkcjonalności kolejek na wyłączność (Exclusive Queue[5]).
- (IV) Kolejka kończąca się na aktualnie podłączonym do maszyny wirtualnej kliencie. Podobnie jak powyżej kolejek istnieje tyle ile aktywnych użytkowników. Celem kolejki jest sprawdzenie, czy aplikacja kliencka nadal jest podłączona do wirtualnej maszyny (mechanizm Exclusive Queue). W celach bezpieczeństwa będą one definiowane na oddzielnym procesie brokera, który będzie można w razie potrzeby udostępnić poza sieć lokalną.

Powyższe 4 grupy kolejek umożliwią prawidłowe działanie systemu. Każdy z modułów tworzy w trakcie uruchamiania kolejki, z których odbiera wiadomości. Jedynym wymogiem prawidłowego uruchomienia komunikacji jest dostępny dla wszystkich serwerów wirtualizacji oraz nadzorców proces brokera.

### 2.2. Zewnętrzne narzędzia

#### 2.2.1. Ansible

Ansible został wykorzystany w systemie do zaaplikowania zmiennej konfiguracji do każdej uruchamianej wirtualnej maszyny. Podstawowo playbook będzie zawierać informacje o:

1. danych dostępowych do dysku sieciowego oraz wykorzystanym protokole,
2. danych dostępowych do usługi katalogowej.

Playbook można rozszerzać o potrzebne dane zależne od użycia.

#### 2.2.2. Vagrant

Vagrant został wykorzystany w celu łatwej parametryzacji oraz powtarzalnego tworzenia maszyn wirtualnych z przygotowanego wcześniej obrazu systemu.

Wykorzystywany jest głównie mechanizm Vagrant-boxów, które są obrazami wcześniej przygotowanego systemu operacyjnego. Aby system działał prawidłowo obraz systemu zamknięty w Vagrantboxie musi spełniać następujące warunki:

1. Użytkownicy muszą być pobierani z usługi katalogowej.
2. Katalogi domowe użytkowników muszą być na dysku sieciowym.
3. Musi istnieć serwer RDP

#### 2.2.3. Libvirt z QEMU

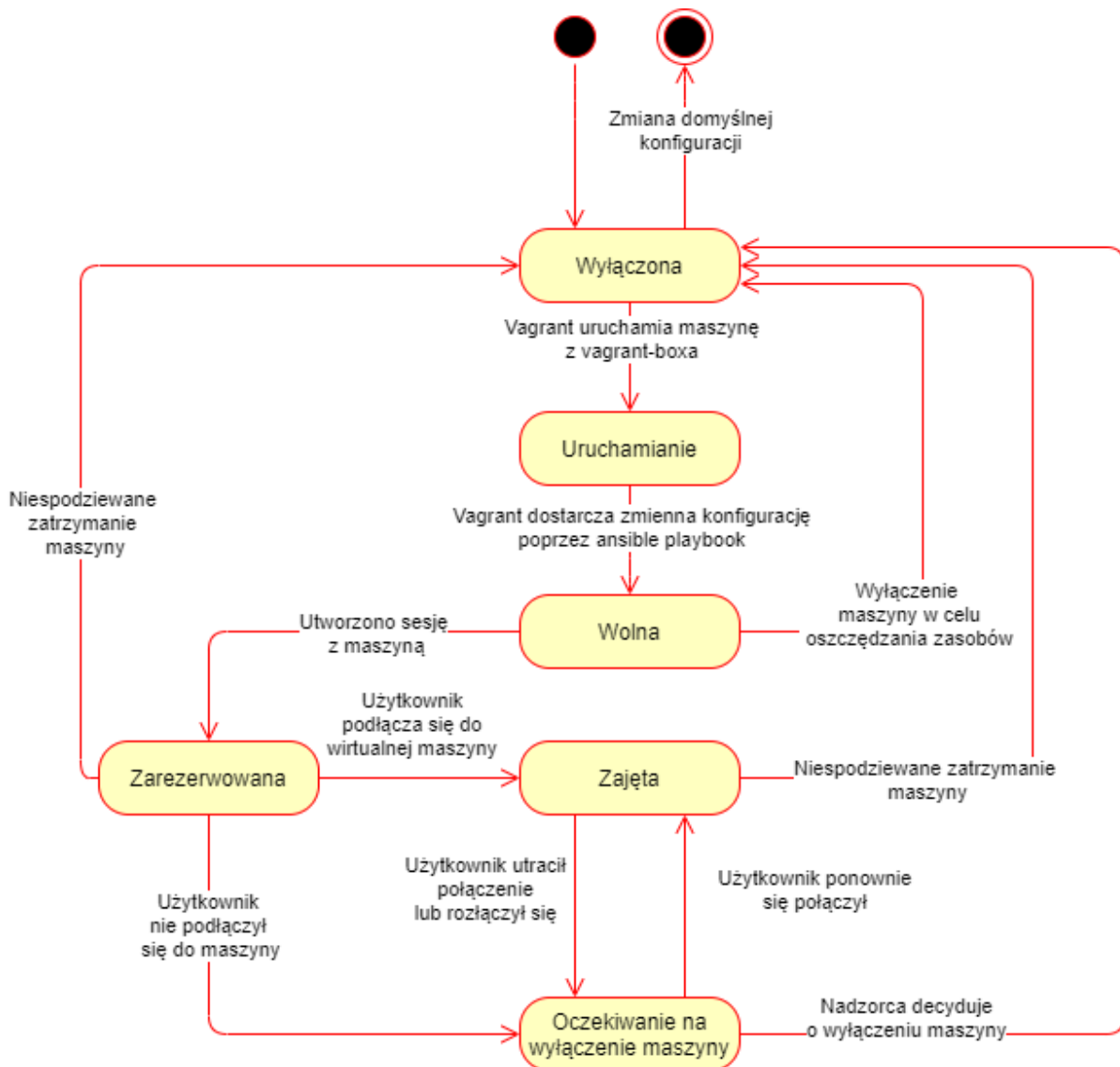
Libvirt połączony z QEMU jest wykorzystany do zarządzania maszynami wirtualnymi uruchamianymi na serwerze wirtualizacji. Umożliwia on:

1. Tworzenie maszyn wirtualnych.
2. Uruchamianie maszyn wirtualnych.
3. Przyporządkowanie zasobów maszynom wirtualnym (w tym krat graficznych).
4. Wyłączanie maszyn wirtualnych.
5. Sprawdzanie, czy maszyna o danej nazwie już działa.

## 2.3. Stany biznesowe

### 2.3.1. Maszyna wirtualna

Najważniejszym obiektem biznesowym w systemie jest maszyna wirtualna, do której będą podłączać się użytkownicy.



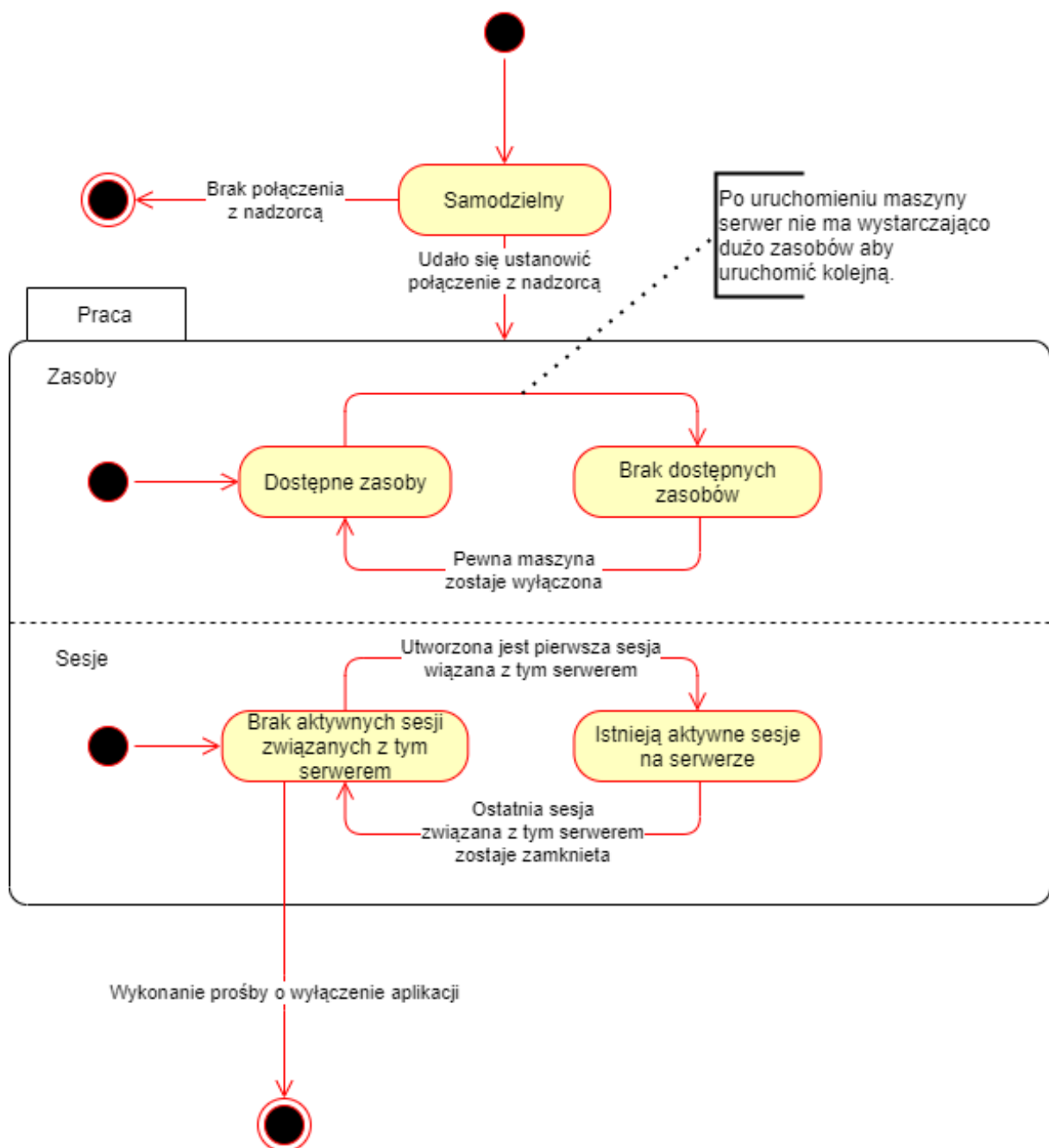
Rysunek 2.3: Diagram stanów dla maszyny wirtualnej

Maszyna, aby być całkowicie uruchomiona, musi zostać zaopatrzona we wszystkie konfiguracje. Stan "Wolna" oznacza możliwość przypisania sesji do tej maszyny. Po przypisaniu maszyny do sesji przechodzi ona w stan oczekiwania na użytkownika. Czas oczekiwania jest konfigurowalny, a po jego upływie maszyna przechodzi w stan oczekiwania na wyłączenie. W tym stanie oczekuje ona na ponowne połączenie, pozwalając użytkownikowi na bezproblemowy powrót do sesji

w przypadku nieoczekiwanego utracenia połączenia. Jeżeli użytkownik nie powróci do sesji, system wyłącza maszynę. Maszyna jest w stanie "Zajęta", gdy aktualnie pracuje na niej użytkownik. Monitorowanie zajętości realizowane jest przy użyciu odpowiedniej kolejki wiadomości.)

### 2.3.2. Serwer wirtualizacji

Serwer wirtualizacji monitoruje zasoby zużywane przez uruchamiane na nim maszyny wirtualne oraz fakt podłączenia do niego użytkowników.



Rysunek 2.4: Diagram stanów dla serwera wirtualizacji

Przy starcie serwer wirtualizacji oczekuje na działającego nadzorcy w sieci. W przypadku jego braku serwer kończy działanie zwracając błąd. Pod względem zasobów, może on mieć wolne zasoby aby utworzyć nowe maszyny, lub też nie. Jednak ważniejszym stanem z perspektywy działania serwera są podłączeni do niego użytkownicy. W przypadku gdy podłączony jest do niego przynajmniej jeden użytkownik, serwer nie może się poprawnie zakończyć pracy aż użytkownik nie skończy używać maszyny.

### 2.3.3. Użytkownik



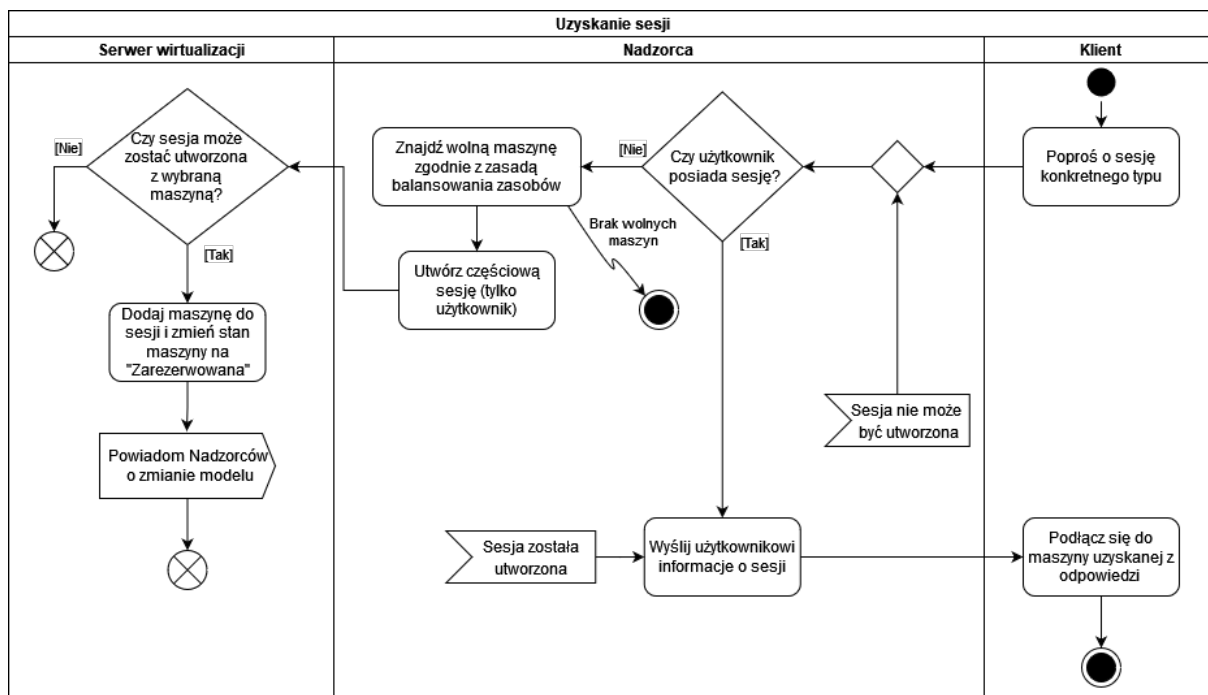
Rysunek 2.5: Diagram stanów dla użytkownika

Użytkownik z perspektywy systemu po zalogowaniu może być 2 dwóch stanach: pracuje w ramach swojej sesji lub też nie. W stanie "Połączony" aplikacja kliencka powiadamia serwer wirtualizacji, że ciągle jest obecny. Przy zmianie stanu do innego informowanie musi ustać, aby serwer mógł wykryć odłączenie się użytkownika.

## 2.4. Procesy biznesowe

### 2.4.1. Uzyskanie sesji

Proces opisuje prośbę klienta o ustanowienie dla niego sesji. Sesja może już istnieć lub zostać utworzona.



Rysunek 2.6: Diagram aktywności dla uzyskania sesji

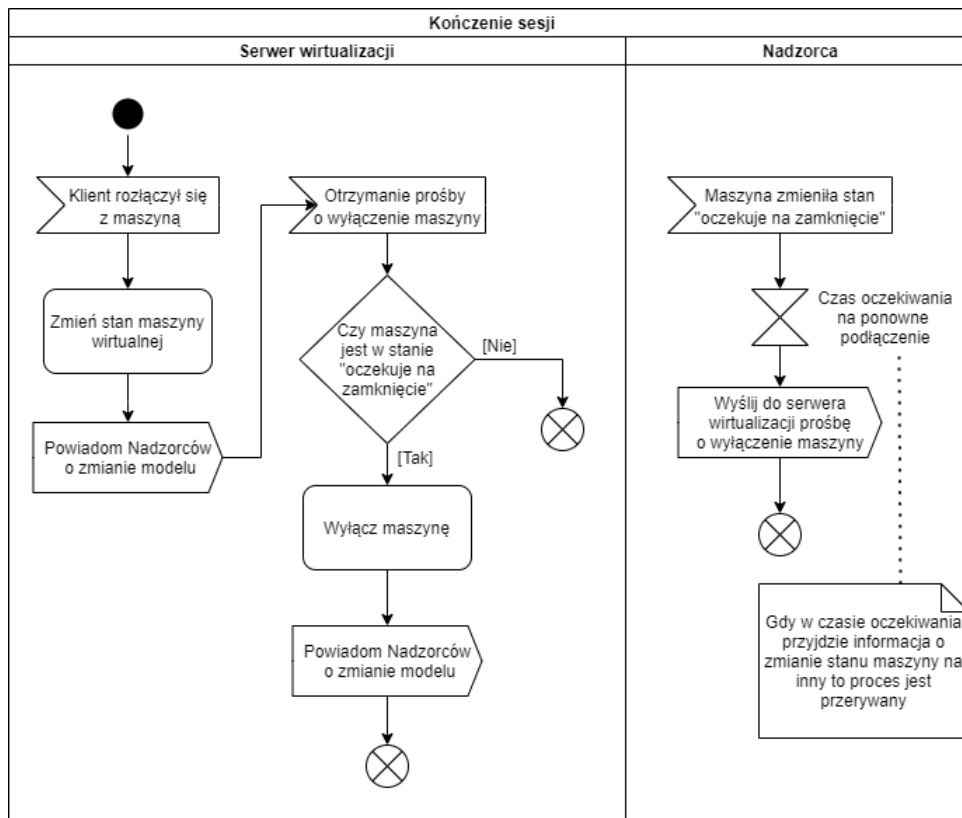
W przypadku gdy sesja już istnieje zostaje ona zwrócona użytkownikowi. W przeciwnym razie nadzorca, na podstawie modelu systemu, wybiera pewną wolną maszynę i wysyła do serwera wirtualizacji, na którym działa wybrana maszyna, prośbę o utworzenie sesji. Możliwość działania wielu nadzorców wymaga, aby proces ten był powtarzalny, czyli dla konkretnego stanu modelu wybrana musi zostać ta sama maszyna. Gdy nie znajdzie wolnej maszyny, to zgłasza błąd do użytkownika. Z założenia taka sytuacja może zajść jedynie, gdy wszystkie maszyny zostały zajęte i brakuje zasobów na utworzenie nowych. Wynika to z tego, że system zawsze powinien trzymać pewien zapas wolnych maszyn. Może się zdarzyć, że model jest nieaktualny i nie można utworzyć sesji z wcześniej wybraną maszyną. Taka prośba zostaje odrzucona przez serwer wirtualizacji, ale zmiana modelu w nadzorcy, wywołana odświeżeniem modelu, spowoduje powtórzenie procesu, tym razem wybierając inną maszynę.

Uzyskanie sesji przez użytkownika zrealizowane jest asynchronicznie. Użytkownik oddzielnym zapytaniem prosi o uzyskanie sesji, po czym używając otrzymanego identyfikatora sesji prosi o jej dane. Obiekt jest w pełni utworzona, gdy odpowiedź nadzorcy sesję w stanie gotowym oraz

adres maszyny do połączenia.

### 2.4.2. Kończenie sesji

Proces ma za zadanie zakończyć sesję oraz wyłączyć skojarzoną z nią wirtualną maszynę w celu zwolnienia zasobów.



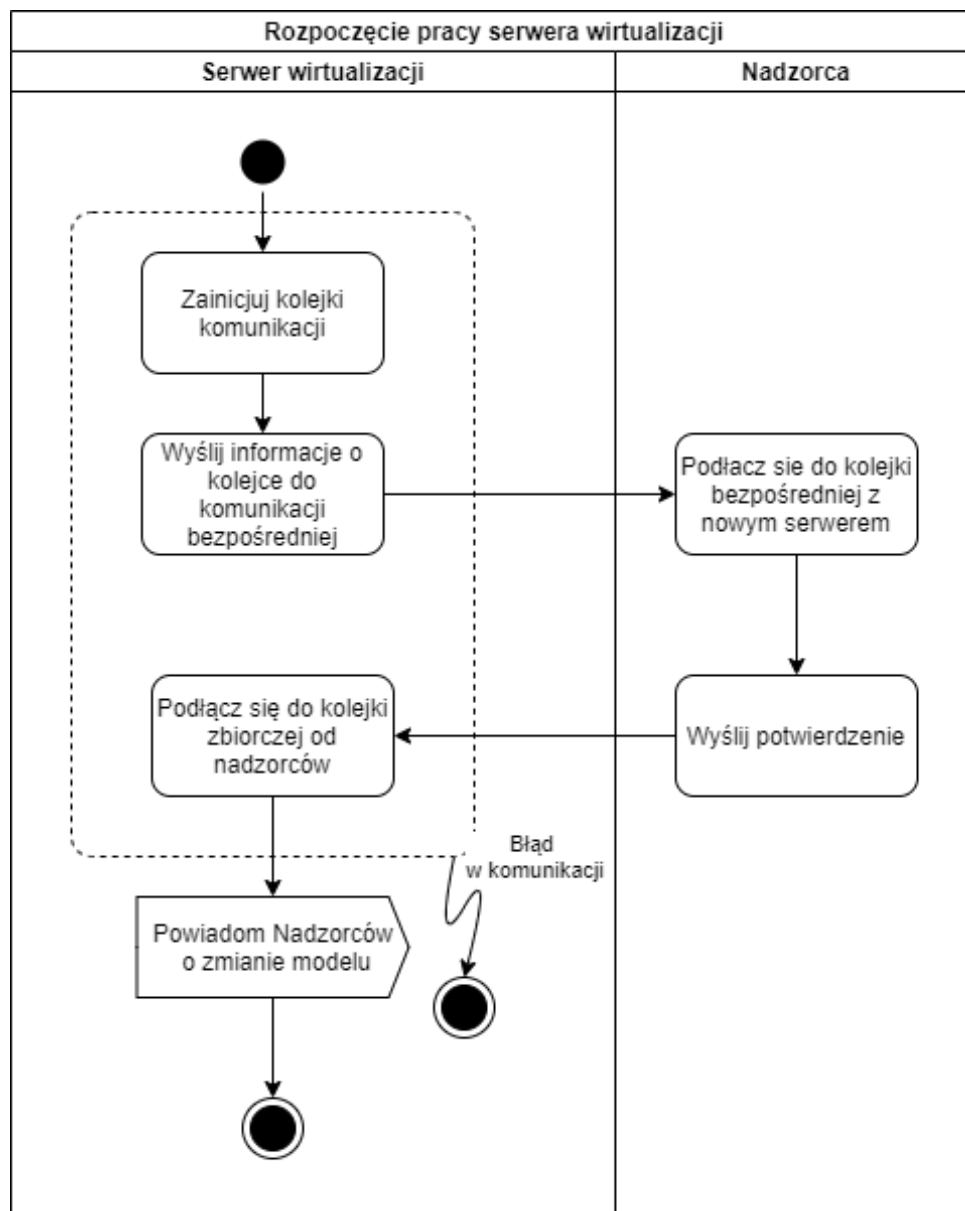
Rysunek 2.7: Diagram aktywności dla zakończenia sesji

Proces rozpoczyna się w momencie, gdy użytkownik odłączy się od systemu lub utraci połączenie. Po upływie ustalonego czasu, jeżeli użytkownik nie podłączył się ponownie, maszyna zostaje wyłączona. Serwer wirtualizacji informuje nadzorcę o utracie połączenia lub odłączeniu się użytkownika poprzez zmianę modelu. Decyzję o wyłączeniu maszyny podejmuje nadzorca. Powoduje to, że zmiana konfiguracji nadzorców będzie oznaczać spójną reakcję całego systemu. Dodatkowo umożliwia to w perspektywie czasu utworzenie bardziej złożonego algorytmu zarządzania zasobami.

### 2.4.3. Rozpoczęcie pracy serwera wirtualizacji

Proces opisuje przyjęcie nowego serwera wirtualizacji do systemu.





Rysunek 2.8: Diagram aktywności dla rozpoczęcia pracy serwera wirtualizacji

Serwer wirtualizacji bez działającego nadzorcy nie jest w stanie obsługiwać użytkowników. Oznacza to, że jeśli przy starcie nie wykryje brokera wiadomości lub nadzorcy po drugiej stronie kolejek[7] to się wyłączy. Jeżeli jednak komunikacja z nadzorcą jest możliwa, to serwer podłączy się do wspólnych kolejek oraz przekaże informacje nadzorcom o kolejce bezpośredniej. Gdy komunikacja będzie ustanowiona bezwarunkowo wyśle stan swojego modelu do nadzorców.

## 2.5. Komunikacja

### 2.5.1. Komunikacja wewnętrzna

Komunikacja wewnątrz systemu opiera się na kolejkach opisanych w opisie modułów. W celu uniknięcia wyścigów i utrzymania spójności modelu systemu pomiędzy nadzorcami ustalone są następujące zasady:

- Nadzorca może zmienić stan systemu jedynie w reakcji na odpowiedź serwera wirtualizacji. Odpowiedzi te wysyłane są do wszystkich nadzorców, dzięki czemu każdy nadzorca ma taki sam model systemu.
- Wiadomości przetwarzane są przez serwer wirtualizacji w sposób atomowy. Pojedyncza wiadomość musi zostać w pełni obsłużona zanim program przejdzie do obsługi kolejnej.
- Serwer wirtualizacji odpowiada na wiadomości wysyłając nowy stan maszyn. Jeżeli żądanie nie może być spełnione z powodu błędnego żądania, to serwer nie odpowiada na żądanie. Wyjątkiem jest żądanie o wysłanie aktualnego stanu maszyn.
- Z powodu asynchroniczności wiadomości moduły nie oczekują na odpowiedź. W przypadku nadzorcy przetwarzanie odpowiedzi zostaje uruchomione przez zmianę modelu.
- Do monitorowania utrzymania połączenia z brokerem użyty jest mechanizm zwracania wiadomości, które nie mogą zostać dostarczone[6]. Używając tego nadzorcy mogą wykryć, kiedy poszczególne serwery wirtualizacji przestaną działać, a serwery wirtualizacji - kiedy wszyscy nadzorcy przestaną działać.

Opisane wyżej założenia pozwalają uniknąć problemu hazardów i wyścigów. Jeżeli wiele nadzorców wyśle do serwera wirtualizacji tą samą prośbę, np. o stworzenie sesji na konkretnej maszynie, to z atomowości obsługi sesja zostanie stworzona tylko dla pierwszego z nich. Serwer wirtualizacji wyśle wiadomość o aktualizacji stanu maszyn i zignoruje pozostałe prośby. Nadzorcy otrzymają zmianę stanów, co spowoduje wywołanie odpowiednich procedur. Dla pierwszego będzie to dalsza część procesu tworzenia sesji, a pozostali nadzorcy pozostaną w procesie wyszukiwania maszyny do sesji.

### 2.5.2. Komunikacja zewnętrzna

Komunikacja aplikacji klienckiej oraz panelu administratora z systemem - nadzorcą - rozwiązana jest za pomocą REST API(<https://restfulapi.net/>). W zależności od konfiguracji

## 2.5. KOMUNIKACJA

nadzorcy wiadomości mogą być wysyłane za pomocą protokołu HTTPS<sup>1</sup>, który zapewnia ich szyfrowanie. W tym celu wymagane jest, aby na adres, pod którym udostępniony będzie system, wystawiony był odpowiedni certyfikat<sup>2</sup>, gwarantujący jego tożsamość.

Całość specyfikacji API umieszczona jest w załączniku. Poniżej znajduje się zestawienie oraz krótki opis endpointów.

login	
POST	/login Log into system
machines	
GET	/machines Get number of available machines grouped into types
session	
POST	/session Get new session of selected type
GET	/session/{sessionId} Get session status
DELETE	/session/{sessionId} Cancel session
resources	
GET	/resources Get servers resources

Rysunek 2.9: Endpointy API

- Login - służy do logowania do systemu; współdzielony przez aplikację kliencką oraz panel administracyjny. Poprawne zalogowanie zwraca token do dalszej autoryzacji.
- Machines - służy do pobierania przez aplikację informacji o typach i ilości dostępnych maszyn. Ten endpoint, oraz wszystkie następne wymagają autoryzacji poprzez umieszczenie tokenu otrzymanego podczas logowania w odpowiednim nagłówku wiadomości, oraz dostępne są tylko dla użytkownika.
- Session - pozwala na wysłanie prośby o uzyskanie sesji, pobranie stanu sesji oraz jej anulowanie. Utworzenie sesji jest możliwe poprzez POST z typem maszyny. W odpowiedzi użytkownik dostaje częściowo wypełniony obiekt sesji zawierający id umożliwiające dalsze zapytania. GET zwraca obiekt sesji z aktualnym stanem. Jeżeli sesja jest gotowa, to zawiera on też adres, z którym należy nawiązać połączenie RDP. DELETE umożliwia anulowanie sesji.
- Resources - udostępnia informację o zasobach działających serwerów wirtualizacji. Dostępny jedynie dla administratora.

<sup>1</sup>Specyfikacja protokołu HTTP Over TLS

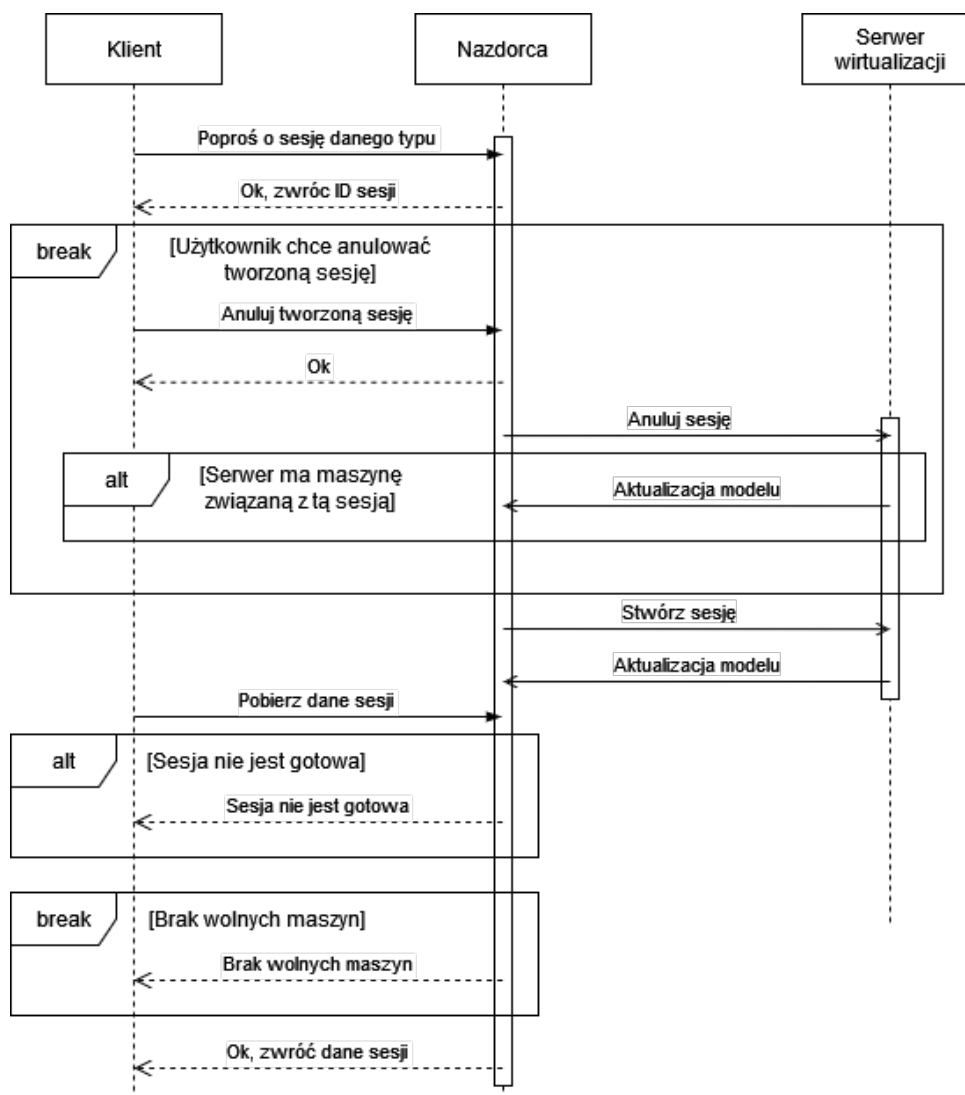
<sup>2</sup>Opis certyfikatu TLS/SSL

Ważną informacją jaką musi posiadać system to fakt, czy użytkownik rzeczywiście jest podłączony do maszyny wirtualnej. System uzyskuje tę informację komunikując się z brokerem wiadomości odpowiedzialnym za komunikację z użytkownikami. Każda aplikacja kliencka po podłączeniu się do maszyny wirtualnej poprzez protokół RDP tworzy kolejkę o takiej nazwie jak uzyskany identyfikator sesji. Serwer wirtualizacji sprawdza co jakiś czas, czy na końcu kolejki istnieje jakikolwiek konsument. Gdy użytkownik się rozłączy to kolejka jest usuwana przez aplikację kliencką, co pozwala serwerowi wykryć odłączenie się użytkownika.

### **2.6. Sekwencje komunikacji**

#### **2.6.1. Utworzenie sesji**

Celem tej sekwencji komunikacji jest odnalezienie istniejącej już sesji lub stworzenie nowej. Zakładamy, że w systemie zawsze jest jakaś wolna maszyna. Inaczej zgłaszamy użytkownikowi błąd.



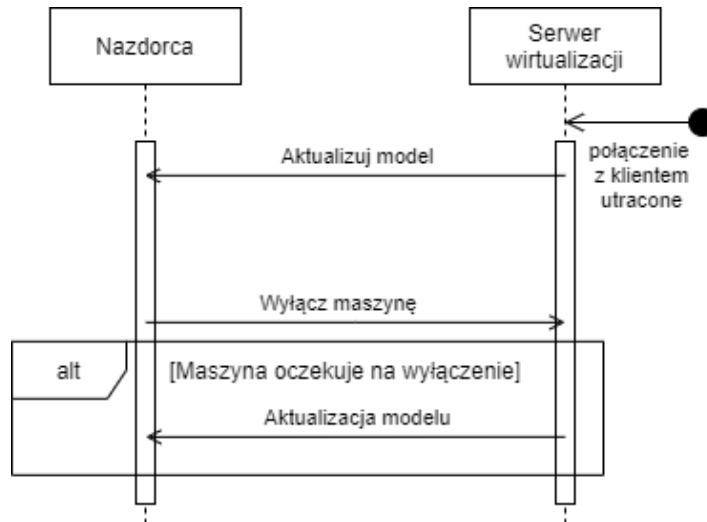
Rysunek 2.10: Sekwencja komunikacji utworzenia sesji

Po prośbie użytkownika nadzorca znajduje wolną maszynę i prosi konkretny serwer wirtualizacji aby spróbował utworzyć sesję dla pewnego użytkownika. Gdy to się uda, ten wysyła zbiorczą kolejką do nadzorców informacje o zmianie modelu. W przeciwnym przypadku nadzorca powtarza wyszukanie wolnej maszyny. O tym, czy nadzorca musi powtórzyć wyszukiwanie decyduje stan otrzymanej maszyny (m.in. czy sesja do niej przypisana należy do tego użytkownika).

Może się zdarzyć także anulowanie wyszukiwania przez użytkownika. Wtedy jeżeli maszyna jest już przydzielona użytkownikowi, to serwer wirtualizacji jest powiadamiany o anulowaniu sesji, aby ją anulował. Jeżeli nie została jeszcze utworzona, to nadzorca dopilnuje aby więcej nie szukać sesji lub wyłączy ją w miarę potrzeby.

### 2.6.2. Zakończenie sesji

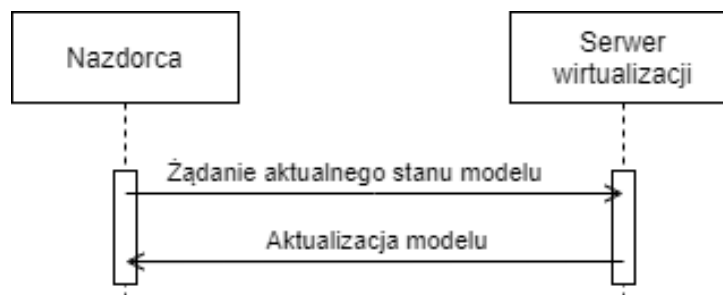
Sekwencja ta zainicjowana jest poprzez utracenie połączenia z użytkownikiem maszyny. Serwer powiadamia o tym fakcie nadzorców, po czym oczekuje na polecenie wyłączenia maszyny przesłane przez nadzorcę. Serwer może odmówić z powodów różnic modelu, lub jeżeli maszyna znów jest używana przez użytkownika, ignorując wiadomość.



Rysunek 2.11: Sekwencja komunikacji zakończenia sesji

### 2.6.3. Aktualizacja stanu

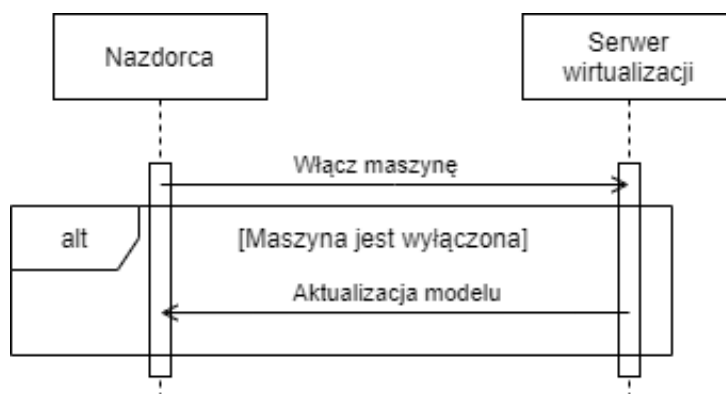
Nadzorca może w każdej chwili poprosić wszystkie serwery wirtualizacji o przesłanie ich aktualnego stanu poprzez wspólną kolejkę do serwerów wirtualizacji. Serwery muszą bezwarunkowo odpowiedzieć aktualnym stanem do wspólnej kolejki zwrotnej.



Rysunek 2.12: Sekwencja komunikacji aktualizacji stanu systemu

### 2.6.4. Włączenie maszyny

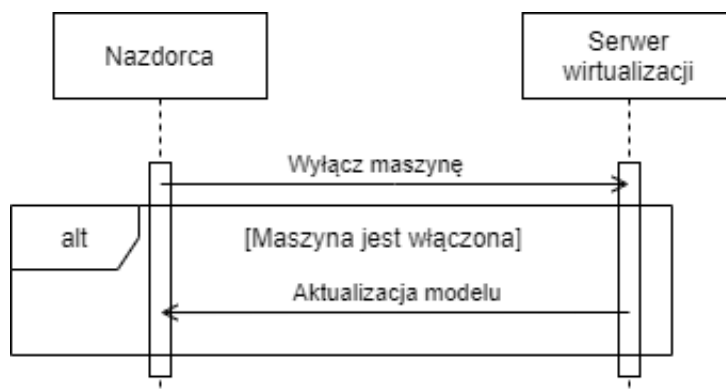
Nadzorca może poprosić konkretny serwer wirtualizacji aby utworzył maszynę o konkretnej nazwie. Jeżeli maszynie nie istnieje, to zostanie uruchomiona oraz serwer odeśle powiadomienie zbiorczą kolejką o zmianie modelu. W przeciwnym wypadku nie zrobi nic.



Rysunek 2.13: Sekwencja komunikacji włączenia maszyny

### 2.6.5. Wyłączenie maszyny

Nadzorca może poprosić konkretny serwer virtualizacji aby wyłączył konkretną maszynę wirtualną. Jeżeli maszynę można wyłączyć to zostanie ona wyłączona. Następnie serwer virtualizacji odeśle powiadomienie zbiorczą kolejką o zmianie modelu. W przeciwnym wypadku nie zrobi nic.



Rysunek 2.14: Sekwencja komunikacji wyłączenia maszyny

## 2.7. Wykorzystane technologie

### 2.7.1. Technologie realizacji strony klienckiej

Realizacja panelu administracyjnego oraz aplikacji klienckiej, przez które użytkownicy i administratorzy korzystają z systemu, zostały wykonane przy użyciu platformy programistycznej Angular. Wykorzystaliśmy ją z powodu znajomości technologii, co umożliwiło sprawną implementację aplikacji. W każdym przypadku skorzystaliśmy z języka TypeScript, który jest wykorzystywany przez Angulara oraz kompiluje się do JavaScriptu.

Aplikacja kliencka uruchamiana jest na systemie operacyjnym użytkownika poprzez techno-

logie Electron, która umożliwia wyświetlanie aplikacji internetowej w postaci okna. W wielkim uproszczeniu Electron korzysta z silnika Chrome'a aby uruchomić aplikację jako stronę internetową i wyświetlić ją jak inne okna w systemie. Ułatwiło nam to stworzenie aplikacji od razu na wiele systemów operacyjnych. Dodatkowym plusem korzystania z Electrona jest możliwość odwołania do silnika NodeJS, aby zwiększyć możliwości dostępu do zasobów systemu operacyjnego. Wykorzystaliśmy go do uruchamiania podprocesu klienta RDP, aby po uzyskaniu sesji od razu zestawić połączenie z wybraną maszyną wirtualną.

### 2.7.2. Technologie realizacji strony serwerowej

W przypadku aplikacji nadzorcy oraz serwera wirtualizacji zdecydowaliśmy się na środowisko uruchomieniowe .NET 5.0 z językiem C#. Ekosystem .NET jest przez nas dobrze poznany i daje nam możliwości aby ułatwić oraz przyspieszyć naszą pracę. Głównym ułatwieniem była duża baza bibliotek zgromadzona w menadżerze pakietów Nuget, do którego dodaliśmy też własne pakiety, aby wydzielić wspólną część kodu aplikacji serwerowych.

Do zarządzania i monitorowania maszyn wirtualnych skorzystaliśmy z libvirta. W serwerze wirtualizacji wykorzystaliśmy rozszerzoną przez nas bibliotekę odwołującą się do oryginalnych bibliotek libvirta. Jest to technologie powszechnie znana i dobrze przez nas poznana. Był to najprostszy sposób do realizacji zarządcy maszyn wirtualnych. Dodatkowo współpracuje ona także z technologią Vagrant, która umożliwia proste powielanie maszyn wirtualnych z wzorca dostarczonego przez użytkownika. Ułatwi ona administratorom tworzenie obrazów uruchamianych systemów operacyjnych.

Serwer wirtualizacji komunikuje się z Vagrantem poprzez specjalnie przygotowany plik wykonywalny z kodem w Ruby. Vagrant wykonywany jest jako podproces serwera wirtualizacji w postaci polecenia powłoki. Parametry przekazywane są poprzez zmienne środowiskowe ustawiane przez serwer wirtualizacji chwilę przed uruchomieniem podprocesu. Do dodatkowych konfiguracji maszyn wirtualnych użyliśmy technologii Ansible. Wybraliśmy ją, ponieważ jest znacznie mniej bezawaryjna niż konfigurowanie systemu przez skrypty powłoki. Oprócz tego skorzystaliśmy z niej do konfiguracji komputerów przed uruchomieniem pisanego przez nas systemu.

### 2.7.3. Technologie testowania

Do testów jednostkowych aplikacji od strony klienckiej skorzystaliśmy z platformy Jest. Ułatwia ona pisanie testów przy niepełnym obrazie systemu. W przypadku rozbudowanej komunikacji naszego systemu było to bardzo przydatne. W przypadku aplikacji serwerowych skorzystaliśmy z platformy NUnit. Zналиśmy ją z wcześniejszych projektów, więc mogliśmy od razu przystąpić



## 2.8. WYMAGANIA SYSTEMU

do pracy. Do testów komunikacji poprzez brokera wiadomości oraz integracji z libvirtem także skorzystaliśmy z NUnita.

Do testów integracyjnych aplikacji Angularowych korzystaliśmy z platformy Cypress. Zdecydowaliśmy się na nią, ponieważ wykonuje ona zrzut widoku podczas każdego kroku testów. Wcześniej znane nam rozwiązania nie posiadały takich funkcji, przez co bardzo trudno było szukać problemów w testach. Dodatkowe testy API wystawianego przez aplikację nadzorcy wykonaliśmy Postmanem.

### 2.7.4. Utrzymanie kodu

Do sprawnego zarządzania projektem wykorzystaliśmy strukturę wielu repozytoriów dostępnych na stronie github. Wspólne części kodu wydzielaliśmy w postaci pakietów NPM oraz Nuget. Aby zautomatyzować proces publikacji pakietów skorzystaliśmy ze środowiska ciągłej integracji Github Actions. Umożliwiło to praktycznie bezproblemową realizację komunikacji pomiędzy nadzorcą a aplikacją kliencką i panelem administratora. Tworzyliśmy automatycznie generowane pakiety z częściową implementacją obiektów transportowych oraz struktury zapytań.

## 2.8. Wymagania systemu

System do poprawnej pracy wymaga konfiguracji środowiska na wielu płaszczyznach. Poniższy rozdział zawiera informacje o konfiguracji systemu, wymaganej do uruchomienia każdego z modułów.

### 2.8.1. Komunikacja sieciowa między modułami

System do poprawnego działania z pełną funkcjonalnością musi składać się z:

1. Przynajmniej jednego nadzorcy (\*).
2. Serwerów wirtualizacji.
3. Maszyn wirtualnych uruchamianych na serwerach wirtualizacji.
4. Serwera HTTP udostępniającego panel administracyjny.
5. Brokera wiadomości do komunikacji wewnętrznej (\*).
6. Brokera wiadomości do komunikacji zewnętrznej (może być tym samym brokerem co wewnętrzny).

7. Dowolnej liczby aplikacji klienckich.

Elementy wymagane do poprawnego uruchomienia oznaczone zostały symbolem (\*).

### **Dostępność wewnętrznego brokera**

Broker wewnętrzny powinien być dostępny dla każdego nadzorcy oraz każdego serwera wirtualizacji. Jest to wymagane do prawidłowej komunikacji pomiędzy nadzorcami a serwerami wirtualizacji.

### **Dostępność zewnętrznego brokera**

Broker zewnętrzny powinien być dostępny dla każdego serwera wirtualizacji oraz dla każdego klienta łączącego się z systemem. Jest to wymagane do stwierdzenia czy użytkownik nadal jest podłączony do maszyny wirtualnej.

### **Dostępność nadzorców**

Nadzorcy powinni być dostępni dla aplikacji klienckich, którzy poprzez nich komunikują się z resztą systemu. Administrator podczas używania panelu administracyjnego z poziomu serwera HTTP powinien móc wysyłać zapytania do nadzorców.

### **Dostępność serwerów wirtualizacji**

Serwery wirtualizacji nie musi być widoczny przez żaden z innych modułów.

### **Dostępność serwera http z panelem administracyjnym**

Serwer HTTP powinien być dostępny dla każdego z administratorów.

### **Dostępność maszyn wirtualnych**

Maszyny wirtualne powinny być dostępne dla każdego z klientów. Jest to potrzebne do pracy na nich poprzez protokół RDP.

#### **2.8.2. Wymagania aplikacji klienckiej**

Aplikacja kliencka wspiera system Windows 10 lub nowszy oraz GNU/Linux w dystrybucjach opartych na 64 bitowych systemach Ubuntu 18.04+ oraz Arch Linux.

Dla systemu Windows aplikacja kliencka, jako plik wykonywalny pobrana ze strony z oficjalnymi wydaniem[4], powinna uruchomić się bez żadnych wcześniejszych konfiguracji. Do uruchomienia wymagany jest jednak plik konfiguracji użytkownika dostępny w oficjalnych wydaniach.

## 2.8. WYMAGANIA SYSTEMU

Do działania korzysta ona z klienta RDP dostarczonego przez firmę Microsoft wraz z systemem Windows. Aplikacja była testowana dla systemu Windows 10. Dla starszych wersji systemu Windows aplikacja może nie działać prawidłowo.

W przypadku systemu Linux należy posiadać środowisko graficzne oraz mieć zainstalowanego klienta FreeRDP (<https://www.freerdp.com/>). Pozostałe zależności są dostarczone wewnątrz pliku `.appimage`. Działanie aplikacji było testowane na dystrybucjach opartych na Debianie oraz Arch Linuxie. Dla innych dystrybucji aplikacja może nie działać prawidłowo.

### 2.8.3. Wymagania aplikacji nadzorcy i serwera panelu administracyjnego

Do uruchomienia aplikacji nadzorcy i serwera HTTP z panelem administracyjnym potrzeba zainstalowanego Dockera w systemie. Każdy z tych dwóch modułów można zbudować do kontenera, w którym wszystkie zależności zostaną spełnione.

Do ręcznego uruchomienia aplikacji nadzorcy, bez pośrednictwa dockera wymagany jest .NET 5. W przypadku panelu administratora do kompilacji wymagany jest NodeJS w wersji 16.13.2. Zbudowany panel można udostępnić za pomocą dowolnego serwera HTTP.

### 2.8.4. Wymagania serwera wirtualizacji

Serwer wirtualizacji, oprócz działającej usługi Dockera, potrzebuje dodatkowych usług. Do prawidłowego działania wymagana jest działająca usługa zarządcy wirtualnych maszyn libvirt w wersji 7.10+. W przypadku ręcznego uruchamiania aplikacji, bez pośrednictwa dockera wymagany jest vagrant w wersji 2.2.11+ oraz .NET 5.

Aby prawidłowo uruchomić maszynę wirtualną potrzebna jest uruchomiona usługa zapory sieciowej oraz pakiet dnsmasq. Wymagane jest także utworzenie struktur usługi vagrant dla użytkownika uruchamiającego serwer wirtualizacji. Chodzi konkretnie o utworzenie folderu `.vagrant.d` w katalogu domowym użytkownika.

Aby uruchamiane maszyny wirtualne były dostępne dla innych urządzeń potrzebne jest utworzenie interfejsu sieciowego w trybie bridge. Nazwa interfejsu powinna być przekazana do pliku konfiguracyjnego serwera wirtualizacji. Maszyny wirtualne uzyskają wtedy dostęp do sieci w taki sposób, jakby były fizycznymi komputerami w sieci.

W niektórych przypadkach podczas uruchamiania maszyny wirtualnej przy użyciu **vagranta** oraz uruchamiania wybranych kontenerów w **dockerze** komunikacja sieciowa z maszyną wirtualną poprzez podłączony interfejs w trybie bridge może zostać ograniczona. Należy wtedy dopilnować by w trakcie działania systemu zapora sieciowa posiadała zasadę na samej górze łańcucha FORWARD, która będzie bezwarunkowo zezwalać trasować pakiety z urządzeń w trybie

bridge. Przy uruchomieniu serwera wirtualizacji w kontenerze dockerowym trzeba dopilnować aby z sieci, w której pracuje kontener, można było osiągnąć sieć konfiguracyjna Vagranta. W przeciwnym wypadku próba uruchomienia maszyny wirtualnej zakończy się z błędem połączenia ssh.

### 2.8.5. Automatyzacja konfiguracji

Przykładowa konfiguracja oraz narzędzia do automatycznej konfiguracji przed uruchomieniem dostępne są w module `configuration`[3]. Składa się on ze skryptów konfiguracyjnych Ansible, nazywanych dalej playbook, oraz zmiennych opisujących konfigurowane komputery.

By uruchomić skrypt dla pewnego systemu operacyjnego, który chcemy skonfigurować, musi on spełniać wymogi opisane w dokumentacji Ansible[14].

#### Grupy i zmienne

Konfiguracja podzielona jest na 2 grupy: `overseer` i `virtsrv`. Odpowiadają one za reprezentację systemów przygotowanych odpowiednio dla nadzorców oraz serwerów wirtualizacji. Dodatkowo wytyczona jest sztuczna grupa `all` opisująca wszystkie konfigurowane systemy.

Jedyną wspólną zmienną dla wszystkich maszyn jest nazwa użytkownika, który będzie uruchamiał i odpowiadał za zasoby systemu OneClickDesktop.

Dla każdej maszyny z osobna należy zdefiniować dane dostępne do komunikacji z konfigurowanym systemem. Dodatkowo należy podać hasło umożliwiające dostęp do uprawnień superużytkownika.

#### Zmienne dla serwera wirtualizacji

Playbook dla serwera wirtualizacji wykona wszystkie kroki opisane w 2.8.4. Aby tego dokonać należy zdefiniować dane dla tworzonego interfejsu typu bridge. Należy zdefiniować nazwę interfejsu sieciowego, który zostanie połączony do nowo tworzonego urządzenia bridge o nazwie zdefiniowanej w pliku. Playbook korzysta z NetworkManagera(<https://networkmanager.dev/>) do zmiany konfiguracji, zatem trzeba podać nazwę *connection* (jednostka logiczna w Network-Managerze) skojarzonego z początkowo wybranym interfejsem sieciowym.

#### Zmienne dla nadzorcy

Aby uruchomić nadzorcę wystarczą wspólne wymagania dla wszystkich grup.

### 2.8.6. Wymagania szablonu maszyny wirtualnej

Maszyna wirtualna uruchamiana w ramach systemu OneClickDesktop musi być w postaci Vagrant Boxa. Przykładowy box został utworzony w czasie rozwoju systemu i dostępny jest w chmurze Vagrant pod nazwą `smogork/archlinux-rdp`[1]. Box używa dystrybucji Arch Linux oraz spełnia wszystkie wymagania aby zostać uruchomionym w ramach systemu.

Do przygotowania szablonu proponujemy aby skorzystać właśnie z tego obrazu. Jeżeli jednak potrzebne jest utworzenie niestandardowego szablonu to musi on:

- Spełniać minimalne wymagania opisane w dokumentacji Vagranta[9].
- Mieć zainstalowany menadżer okienek. Proponowany menadżer okienek, zawarty w przykładowym szablonie, to XFCE w wersji 4.16.
- Przy uruchomieniu udostępniać usługę zdalnego pulpitu RDP. Przykładowy szablon korzysta z implementacji xRDP w wersji 0.9.17-1.
- Każdy nowy interfejs sieciowy powinien być tak skonfigurowany, aby uzyskać adres IP z usługi DHCP.
- Przy starcie systemu uruchamiać usługę `qemu-guest-agent`.

Nie ma ograniczenia co do systemu operacyjnego uruchamianego wewnątrz systemu OneClickDesktop, tak długo jak jest on w stanie spełnić powyższe wymagania.

Aby zbudować własny szablon należy skonsultować się z dokumentacją wtyczki `vagrant-libvirt`[12].

## 2.9. Uruchomienie systemu

W tym podrozdziale zakładamy, że każdy system operacyjny został skonfigurowany poprawnie zgodnie z opisem w 2.8. Wtedy można przejść do uruchamiania systemu.

### 2.9.1. Pozyskanie aplikacji klienckiej

Zalecanym sposobem pozyskania aplikacji klienckiej jest udanie się do sekcji z oficjalnymi wydaniem[4] modułu aplikacji klienckiej oraz pobranie najnowszej wersji dla wybranego systemu operacyjnego. Przy pierwszym uruchomieniu trzeba pobrać archiwum plików zawierające aplikację oraz plik konfiguracyjny. Bez pliku konfiguracyjnego aplikacja nie uruchomi się.

### 2.9.2. Zbudowanie kontenerów

Moduły: panelu administracyjnego, nadzorcy i serwera wirtualizacji można uruchomić pod postacią kontenera Dockerowego. Jest to zalecany sposób uruchamiania tych trzech modułów.

#### Ujednolicony sposób budowania kontenerów

Każdy z tych 3 modułów ma przygotowany skrypt `build.sh`, który powinien prawidłowo zbudować kontener. Skrypt ten wykona polecenie do budowania i oznaczy kontener odpowiednią nazwą. Należy wykonać go zawsze z poziomu głównego folderu repozytorium modułu. Każdy kontener przy starcie wykona skrypt `assets/entry_point.sh`.

#### Budowanie kontenera serwera wirtualizacji

Kontener serwera wirtualizacji wymaga specjalnie przygotowanego wcześniej kontenera zawierającego wszystkie potrzebne zależności do uruchomienia aplikacji. Aby go zbudować należy skorzystać z pliku `runtime_container/Dockerfile` i oznaczyć go nazwą `click-desktop/virtualization-server-runtime`. Kontener zawierający aplikację w czasie budowania będzie oczekiwał, że taki obraz istnieje.

Po zbudowaniu kontenera z zależnościami można przystąpić do zbudowania kontenera głównego. Należy do tego celu skorzystać z dostarczonego pliku `Dockerfile` w głównym folderze repozytorium.

Przy uruchomieniu skryptu `build.sh` kontener jest budowany z nazwą `one-click-desktop/virtualization-server`.

#### Budowanie kontenera nadzorcy i panelu administratora

W przypadku tych dwóch modułów nie trzeba wykonać żadnych dodatkowych przygotowań. Wystarczy skorzystać z dostarczonego pliku `Dockerfile` w głównym folderze repozytorium.

Przy uruchomieniu skryptu `build.sh` kontenery są budowane z nazwami odpowiednio `one-click-desktop/overseer` oraz `one-click-desktop/admin-panel`.

### 2.9.3. Budowanie modułów

Każdy z modułów posiada dokładną instrukcję budowania w pliku `README.md`. Uruchamianie zbudowanych modułów poleca się tylko w przypadku rozwoju aplikacji. Przy wdrażaniu systemu najszybciej i najbezpieczniej jest skorzystać z metod opisanych w 2.9.1 i 2.9.2.

### Moduły napisane w technologii .NET

W przypadku budowania modułu nadzorcy i serwera wirtualizacji potrzebne są narzędzia deweloperskie .NET 5.0. W trakcie budowania aplikacji wszystkie użyte biblioteki zostaną pobrane przy użyciu menadżera pakietów Nuget. Przy uruchomieniu serwera wirtualizacji potrzebny jest, poza wymaganiami zdefiniowanymi w 2.8.4, zainstalowany vagrant wraz z wtyczką vagrant-libvirt w wersji przynajmniej 0.7.0.

### Moduły napisane w technologii Node

W przypadku budowania modułów aplikacji klienckiej oraz panelu administracyjnego potrzebny będzie pakiet Node. Przed zbudowaniem aplikacji należy pobrać wszystkie użyte biblioteki przy użyciu menadżera pakietów npm. Są one zdefiniowane w projekcie aplikacji i zostaną pobrane automatycznie.

#### 2.9.4. Konfiguracja aplikacji klienckiej

Aplikacja kliencka wymaga pliku konfiguracyjnego o nazwie `config.json` w tym samym folderze co plik wykonywalny. W pliku konfiguracyjnym użytkownik musi podać:

- Adres jednego z nadzorców (`basePath`) - adres musi być w formacie URI.
- Adres zewnętrznego brokera wiadomości (`rabbitPath`) - adres musi być w formacie URI.
- Czy aplikacja powinna podczas połączenia RDP używać danych dostępowych takich samych jak przy logowaniu do systemu (`useRdpCredentials`) - `true` albo `false`.
- Czy aplikacja powinna uruchamiać zintegrowanego klienta RDP (`startRdp`) - `true` albo `false`. Przydaje się to przy wykorzystaniu innego niż zalecany klient RDP. Aplikacja po uzyskaniu sesji wyświetli dane dostępowe do przypisanej maszyny wirtualnej.

#### 2.9.5. Konfiguracja panelu administracyjnego

Przy uruchomieniu panelu administracyjnego trzeba przekazać adres jednego z nadzorców. Aby tego dokonać należy ustawić zmienną środowiskową `API_URL` na adres dostępowy jednego z nadzorców.

### 2.9.6. Konfiguracja nadzorcy

Nadzorca posiada wiele parametrów związanych z działaniem całego systemu. Kontroluje on między innymi czas oczekiwania na powrót użytkownika do porzuconej sesji. Wszystkie parametry należy przekazać mu poprzez plik konfiguracyjny.

Nazwa pliku konfiguracyjnego musi spełniać wzorzec `appsettings.${Nazwa_srodowiska}.ini`. W przypadku uruchamiania aplikacji wewnątrz kontenera ustawione jest środowisko o nazwie `Production`. Aplikacja nadzorcy domyślnie przeszukuje folder `./config` w poszukiwaniu plików konfiguracyjnych. Można zmienić ścieżkę do folderu konfiguracyjnego poprzez parametr uruchomienia `-c path/to/other/config/folder/`.

W pliku konfiguracyjnym nadzorcy poszukuje 2 specjalnych sekcji:

- `JwtSettings` - przechowuje parametr wykorzystywany do generowania unikatowych tokenów autoryzacyjnych
- `OneClickDesktop` - przechowuje parametry związane z działaniem systemu `OneClickDesktop`.

Poza tymi sekcjami można tam umieścić dowolne sekcje, które będzie przetwarzać ASP.NET. Jednak warte uwagi są następujące sekcje:

- `Logging` - parametry loggera dostarczonego przez Microsoft(<https://www.nuget.org/packages/microsoft.extensions.logging>).
- `Kestrel` - parametry serwera HTTP udostępniającego aplikację. W wypadku aplikacji nadzorcy bez ustawienia certyfikatu nie zadziała ona w trybie HTTPS.

Pozostają jeszcze 2 ważne parametry bez sekcji zaimplementowane przez ASP.NET:

- `AllowedHosts` - lista adresów z których zapytania będą obsługiwane.
- `urls` - lista adresów na których aplikacja będzie nasłuchiwać zapytań.

#### Parametry sekcji `JwtSettings`

Sekcja ta zawiera jedynie parametr `Secret`. Należy go ustawić na dowolny napis.

#### Parametry sekcji `OneClickDesktop`

Sekcja zawiera parametry związane z komunikacją pomiędzy jednostkami systemu oraz kilka parametrów określających interwały czasowe zdarzeń. W dostarczonej przykładowej konfiguracji wykomentowane wartości oznaczają wartości domyślne.



## 2.9. URUCHOMIENIE SYSTEMU

- **OverseerId** - identyfikator nadzorcy używany w komunikacji poprzez wewnętrznego brokera wiadomości. Powinna być unikatowa w skali jednej instancji systemu OneClickDesktop. Domyślnie przyjmuje wartość **overseer-test**.
- **RabbitMQHostname** - adres dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość **localhost**. Bez działającego procesu brokera pod tym adresem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **RabbitMQPort** - port dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość **5672**. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ModelUpdateInterval** - ilość sekund co ile nadzorca prosi serwery wirtualizacji o aktualizacje modelu. Domyślnie przyjmuje wartość **60**. Zalecane jest aby wartość tego parametru opisywała czas od 30 do 60 sekund.
- **DomainShutdownTimeout** - ilość minut ile musi upłynąć od oznaczenia maszyny wirtualnej stanem **Oczekiwanie na wyłączenie maszyny** do jej wyłączenia. Domyślnie przyjmuje wartość **15**.
- **DomainShutdownCounterInterval** - ilość sekund co ile nadzorca sprawdza czy maszyna wirtualna nadal oczekuje na zamknięcie. Zaleca się, aby ten czas dzielił czas z parametru **DomainShutdownTimeout** na równe części. Takich części nie powinno być mniej niż 10, ale także nie więcej niż 100. Każde takie sprawdzenie zużywa czas procesora.

### Przykładowa konfiguracja

Konfiguracja ta umożliwia użycie protokołu HTTPS dzięki przekazaniu zabezpieczonego hasłem certyfikatu oraz nasłuchiowaniu na adresie obsługującym ten protokół.

```
AllowedHosts=*
```

```
urls=http://*:5000;https://*:5001
```

```
[Kestrel:Certificates:Default]
```

```
Password=password
```

```
Path=/overseer/overseer.pfx
```

```
[JwtSettings]
```

```
Secret=MySecretForJWTTokensPleaseChangeMe
```

```
[Logging.LogLevel]
Default=Information
Microsoft=Warning
Microsoft.Hosting.Lifetime=Information
```

```
[OneClickDesktop]
OverseerId=overseer-instance-123
RabbitMQHostname=1.2.3.4
RabbitMQPort=5678
ModelUpdateInterval=60
DomainShutdownTimeout=15
DomainShutdownCounterInterval=30
```

### 2.9.7. Konfiguracja serwera wirtualizacji

Serwer wirtualizacji posiada wiele parametrów związanych z zasobami przekazanymi do dyspozycji systemu. Wszystkie parametry należy przekazać mu poprzez pliki konfiguracyjne.

Pliki te muszą znaleźć się w jednym folderze. Głównym plikiem konfiguracyjnym jest `virtsrv.ini`. Zawiera on parametry związane z komunikacją wewnątrz i na zewnątrz systemu. Dodatkowo znajdują się tam informacje o udostępnionych zasobach dla serwera wirtualizacji. Dodatkowe pliki konfiguracyjne reprezentują typy maszyn wirtualnych uruchamianych na tym serwerze wirtualizacji. Jedynym odstępstwem od reguły jest plik konfiguracyjny dla pakietu NLog. Musi on się znajdować w tym samym folderze co aplikacja oraz nazywać się `NLog.config`.

Aplikacja serwera wirtualizacji domyślnie przeszukuje folder `./config` w poszukiwaniu plików konfiguracyjnych. Można zmienić ścieżkę do folderu konfiguracyjnego poprzez parametr uruchomienia `-cpath/to/other/config/folder/`.

### Główny plik konfiguracyjny

W głównym pliku konfiguracyjnym możemy wyróżnić 2 sekcje:

- **OneClickDesktop** - przechowuje parametry związane z działaniem systemu OneClickDesktop.
- **ServerResources** - przechowuje parametry określające zgrubnie wszystkie udostępnione zasoby.

W sekcji `OneClickDesktop` występują parametry:

- `VirtualizationServerId` - identyfikator serwera wirtualizacji używany w komunikacji poprzez wewnętrznego brokera wiadomości. Powinna być unikatowa w skali jednej instancji systemu `OneClickDesktop`. Domyślnie przyjmuje wartość `virtsrv-test`.
- `OversserserCommunicationShutdownTimeout` - po upływie tylu sekund bez komunikacji od jakiegokolwiek nadzorcy serwer wirtualizacji uznaje, że zabrakło nadzorców w systemie. Nastąpi wtedy wyłączenie aplikacji serwera wirtualizacji. Domyślnie parametr przyjmuje wartość 120. Zaleca się aby wartość parametru była większa niż dwukrotność parametru `ModelUpdateInterval` z konfiguracji nadzorcy opisanej w 2.9.6.
- `LibvirtUri` - adres do połączenia z usługą libvirta. Vagrant jak i biblioteka libvirta skorzysta z tego adresu do połączenia. Specyfikacja libvirta dokładnie opisuje format tego adresu[11].
- `VagrantFilePath` - ścieżka do specjalnie przygotowanego pliku wsadowego dla Vagranta. Wykorzystywany jest przez system do uruchamiania i wyłączania maszyn wirtualnych. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- `PostStartupPlaybook` - skrypt wykonywany przy pomocy Ansible'a zaraz po uruchomieniu maszyny wirtualnej. Koniecznie należy przetestować czy konfiguracja wykonuje się prawidłowo. Przy każdym błędzie wykonania skryptu maszyna wirtualna zostanie usunięta. Domyślnie przyjmuje wartość `res/poststartup_playbook.yml`.
- `VagrantboxUri` - identyfikator szablonowej maszyny wirtualnej. Musi ona spełniać szereg wymogów opisanych w 2.8.6. W przeciwnym wypadku system nie będzie w stanie uruchomić takiego szablonu. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- `BridgeInterfaceName` - nazwa interfejsu sieciowego w skonfigurowanego w trybie bridge, do którego zostanie podłączona każda uruchomiona maszyna wirtualna. Domyślnie przyjmuje wartość `br0`.
- `BridgedNetwork` - adres sieci, do której podłączona zostanie maszyna wirtualna poprzez `BridgeInterfaceName`, podany w formacie CIDR. Jeżeli maszyna nie będzie posiadać adresu z podanej sieci po uruchomieniu to zostanie wyłączona. Aplikacja nie zadziała prawidłowo bez ustawionej wartości parametru.
- `InternalRabbitMQHostname` - adres dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość `localhost`. Bez działającego procesu brokera pod tym adresem

aplikacja wyłączy się z błędem zaraz po uruchomieniu.

- **InternalRabbitMQPort** - port dostępowy wewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość 5672. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ExternalRabbitMQHostname** - adres dostępowy zewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość `localhost`. Bez działającego procesu brokera pod tym adresem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ExternalRabbitMQPort** - port dostępowy zewnętrznego brokera wiadomości. Domyślnie przyjmuje wartość 5673. Bez działającego procesu brokera pod tym portem aplikacja wyłączy się z błędem zaraz po uruchomieniu.
- **ClientHeartbeatChecksForMissing** - liczba nieudanych sprawdzeń czy aplikacja kliencka nadal jest połączona do maszyny wirtualnej. Po tej liczbie prób maszyna wirtualna oznaczana jest jako oczekująca na zamknięcie. Domyślnie przyjmuje wartość 2.
- **ClientHeartbeatChecksDelay** - czas w milisekundach pomiędzy sprawdzeniami, czy aplikacja kliencka nadal jest połączona do maszyny wirtualnej. Domyślnie przyjmuje wartość 10000

W sekcji **ServerResources** występują parametry:

- **Cpus** - liczba wątków które może wykorzystać system. Domyślnie przyjmuje wartość 2.
- **Memory** - ilość pamięci operacyjnej w MiB jaką może wykorzystać system. Domyślnie przyjmuje wartość 2048.
- **Storage** - ilość przestrzeni dyskowej w GiB jaką może wykorzystać system. Domyślnie przyjmuje wartość 100.
- **GPUsCount** - ilość kart graficznych przekazanych do systemu. Domyślnie przyjmuje wartość 0.
- **MachineTypes** - lista typów maszyn wirtualnych. Każda nazwa typu jest oddzielona od sąsiednich przecinkiem. Nazwa typu musi składać się ze znaków opisanych następującym wyrażeniem regularnym `[a-zA-Z0-9\_-]`. Dla każdej z nazw wyszukiwany jest plik konfiguracyjny o nazwie zaczynającej się nazwą typu i kończącej się `_template.ini`.

## 2.9. URUCHOMIENIE SYSTEMU

Gdy  $n = \text{GPUsCount}$  jest większy od 0, wtedy w pliku muszą znaleźć się sekcje od `ServerGPU.1` do `ServerGPU.n` włącznie. W przeciwnym wypadku serwer wirtualizacji zakończy się z błędem zaraz po uruchomieniu.

Każda z tych sekcji zawiera parametr `AddressCount`, który określa jak duża jest grupa IOMMU w której znajduje się przekazywane urządzenie PCI. W zależności  $m = \text{AddressCount}$  w tej sekcji muszą znaleźć się parametry od `Address_1` do `Address_m` włącznie. W przeciwnym wypadku serwer wirtualizacji zakończy się z błędem zaraz po uruchomieniu. Każdy z tych parametrów opisuje adres pojedynczego urządzenia na magistrali PCI. Adres na magistrali PCI musi zostać opisany w formacie uzyskanym z polecenia `lspci -{domain:4}:{bus:2}:{slot:2}.{function:1}`.

Przykładowa zawartość głównego pliku konfiguracyjnego:

```
[OneClickDesktop]
VirtualizationServerId=virtsrv-instance-123
OverssersCommunicationShutdownTimeout=120
LibvirtUri=qemu:///system
VagrantFilePath=res/Vagrantfile
PostStartupPlaybook=res/poststartup_playbook.yml
VagrantboxUri=smogork/archlinux-rdp

BridgeInterfaceName=br0
BridgedNetwork=1.2.3.0/24

InternalRabbitMQHostname=1.2.3.4
InternalRabbitMQPort=5678

ExternalRabbitMQHostname=1.2.3.4
ExternalRabbitMQPort=8765

ClientHeartbeatChecksForMissing=2
ClientHeartbeatChecksDelay=10000

[ServerResources]
Cpus=6
Memory=4096
```

```
Storage=200
GPUsCount=1
MachineTypes=cpu,cpu-memory
```

```
[ServerGPU.1]
AddressCount=2
Address_1=0000:03:00.0
Address_2=0000:03:00.1
```

### Pliki konfiguracyjne opisujące typy maszyn wirtualnych

W pliku opisującym typy zawarte jest ile potrzeba zasobów aby uruchomić maszynę wytworzoną z tego typu.

Każdy typ ma przypisaną nazwę. Oznaczmy ją jako `${template_name}`. Nazwa typu powinna być unikatowa na skale systemu. Jeżeli chcemy udostępnić ten sam typ na wielu serwerach wirtualizacja należy nadać mu takie same zasoby. System może nieprawidłowo zarządzać zasobami, gdy uzyska 2 wzorce zasobów tego samego typu maszyny. Plik opisujący typ musi mieć nazwę `${template_name}_template.ini`. Plik konfiguracyjny dla wybranego typu musi znajdować się w folderze z pozostałymi plikami konfiguracyjnymi. Jeżeli dla jakiegokolwiek nazwy typu aplikacja nie znajdzie pliku konfiguracyjnego to zakończy się z błędem zaraz po uruchomieniu.

W znalezionym pliku musi być sekcja o nazwie `${template_name}_template`. Zwiera ona parametry:

- **HumanReadableName** - nazwa reprezentowana w aplikacji klienckiej dla użytkownika. Domyślnie przyjmuje wartość `${template_name}`.
- **Cpus** - liczba wątków potrzebna do uruchomienia maszyny tego typu. Domyślnie parametr przyjmuje wartość 2.
- **Memory** - ilość pamięci operacyjnej w MiB potrzebnej do uruchomienia maszyny tego typu. Domyślnie parametr przyjmuje wartość 512.
- **Storage** - ilość przestrzeni dyskowej w GiB potrzebnej do uruchomienia maszyny tego typu. Domyślnie parametr przyjmuje wartość 20.
- **AttachGpu** - informacja czy maszyna tego typu przy uruchomieniu powinna mieć przekazaną kartę graficzną. Domyślnie parametr przyjmuje wartość `false`.

Przykładowa zawartość pliku konfiguracyjnego typu maszyny:

## 2.9. URUCHOMIENIE SYSTEMU

```
[gpu_template]
```

```
HumanReadableName=Efficiency machine with GPU attached
```

```
Cpus=1
```

```
Memory=2048
```

```
Storage=55
```

```
AttachGpu = true
```

### 2.9.8. Procedura uruchomienia

Prawidłowy start systemu powinien zachować następującą kolejność:

1. Uruchomić zewnętrznego i wewnętrznego brokera wiadomości.
2. Poczekać na prawidłowy start brokerów wiadomości.
3. Uruchomić wymaganą liczbę nadzorców.
4. Poczekać na prawidłowy start przynajmniej jednego z nadzorców.
5. uruchomić serwer HTTP udostępniający panel administracyjny.
6. Uruchomić wszystkie serwery wirtualizacji.
7. Poczekać aż w systemie znajdą się w pełni uruchomione maszyny wszystkich zarejestrowanych typów.

Po opisanych krokach system jest gotowy do podłączenia się przez użytkowników. Ważne jest aby przed uruchomieniem jakiegokolwiek nadzorcy brokery wiadomości były już prawidłowo zainicjalizowane. W przeciwnym wypadku aplikacja nadzorcy zakończy się z błędem. Serwer wirtualizacji także zakończy się z błędem, jeżeli zabraknie brokerów wiadomości. Dodatkowo, jeżeli nie będzie żadnego nadzorcy nasłuchującego poprzez wewnętrznego brokera wiadomości, serwer wirtualizacji zgłosi błąd i zakończy pracę.

### 2.9.9. Parametry uruchomienia kontenera panelu administracyjnego

Kontener zawierający panel administracyjny udostępnia aplikacje poprzez serwer HTTP nginx. Do komunikacji wystawia on porty 80 (HTTP) oraz 443 (HTTPS). Należy przekierować je na odpowiednie porty uruchamiającego systemu.

## Adres dostępu do nadzorców

Aplikacja panelu administracyjnego oczekuje ustalonego adresu dostępu do jakiegokolwiek nadzorcy. Przy uruchomieniu kontenera należy ustawić zmienną środowiskową `API_URL` zgodnie z opisem w 2.9.5.

## Certyfikat SSL

W celu uruchomienia panelu w trybie HTTPS należy przekazać zmienioną konfigurację serwera nginx oraz parę klucza z certyfikatem pod postacią woluminu.

```
-v {PATH_TO_CERT}:{PATH_TO_CERT_IN_CONTAINER}
-v {PATH_TO_KEY}:{PATH_TO_KEY_IN_CONTAINER}
-v {PATH_TO_CONF}:/etc/nginx/conf.d/default.conf
```

gdzie `PATH_TO_CERT`, `PATH_TO_KEY` i `PATH_TO_CONF` są ścieżkami bezwzględnymi na systemie uruchamiającym, a `PATH_TO_CERT_IN_CONTAINER` oraz `PATH_TO_KEY_IN_CONTAINER` są ścieżkami bezwzględnymi w kontenerze podanymi w przyłączonej konfiguracji.

### 2.9.10. Parametry uruchomienia kontenera nadzorcy

Aplikacja nadzorcy udostępnia API poprzez serwer HTTP Kestrel. Do komunikacji wystawia porty 5000(HTTP) oraz 5001(HTTPS), które można zmienić w dostarczonej konfiguracji. Należy przekierować je na odpowiednie porty uruchamiającego systemu.

## Plik konfiguracyjny

Aplikacja nadzorcy poszukuje plików konfiguracyjny w lokalizacji `/overseer/config/`. Należy przekazać folder zawierający konfiguracje z systemu uruchamiającego(`PATH_TO_CONFIGS`) do wnętrza kontenera pod dokładnie tą lokalizację pod postacią woluminu.

```
-v {PATH_TO_CONFIGS}:/overseer/config
```

## Certyfikat SSL

W celu uruchomienia nadzorcy w trybie HTTPS należy przekazać certyfikat SSL w formie `.pfx` przy pomocy woluminów oraz odpowiednio przygotowany plik konfiguracyjny tak jak opisano w 2.9.6. Gdy zabraknie certyfikatów a konfiguracja pokazuje, że ma się rozpocząć nasłuchiwanie w trybie HTTPS, nadzorca zakończy się z błędem zaraz po uruchomieniu.

```
-v {PATH_TO_CERT}:{PATH_TO_CERT_IN_CONTAINER}
```



## 2.9. URUCHOMIENIE SYSTEMU

gdzie `PATH_TO_CERT` jest ścieżką bezwzględną do certyfikatu na systemie uruchamiającym oraz `PATH_TO_CERT_IN_CONTAINER` jest ścieżką bezwzględną wewnątrz kontenera podaną w konfiguracji.

### 2.9.11. Parametry uruchomienia kontenera serwera wirtualizacji

Serwer wirtualizacji zarządza maszynami wirtualnymi na systemie uruchamiającym. Wymaga dość nietypowo podłączonych woluminów aby prawidłowo funkcjonować.

#### Zasoby libvirta

Aby komunikować się z usługą libvirta uruchomioną na systemie uruchamiającym potrzebujemy gniazda sieciowego przekazanego do wnętrza kontenera. Musi ono udawać, że jest uruchomiona usługa libvirta we wnętrzu kontenera. Dodatkowo nowo tworzone maszyny powinny zapisywać się pomiędzy uruchomieniami kontenera. W tym celu trzeba przekazać także cały folder z danymi libvirta. Tai efekt można uzyskać przy pomocy woluminów przekazując

```
-v /var/run/libvirt/libvirt-sock:/var/run/libvirt/libvirt-sock  
-v /var/lib/libvirt:/var/lib/libvirt/
```

#### Zasoby vagranta

Przy starcie kontenera zasoby vagranta są puste. Oznacza to, że przy każdym pierwszym uruchomieniu maszyny wirtualnej będzie ona musiała być pobrana i rozpakowana do zasobów libvirta. Zabiera to czas i przestrzeń dyskową. Aby zapewnić trwałość Vagrant Boxów pomiędzy uruchomieniami oraz zarządzanie nimi z poziomu systemu uruchamiającego należy przekazać do kontenera główny folder vagranta. Folder domowy użytkownika wykonawczego powinien być wykorzystany do tego celu.

```
-v ${HOME}/.vagrant.d/boxes:/root/.vagrant.d/boxes/
```

gdzie `HOME` to ścieżka do folderu domowego użytkownika wykonawczego.

#### Plik konfiguracyjny

Aplikacja serwera wirtualizacji poszukuje plików konfiguracyjny w lokalizacji `/app/config/docker-test/`. Należy przekazać folder zawierający konfiguracje z systemu uruchamiającego(`PATH_TO_CONFIGS`) do wnętrza kontenera pod dokładnie tą lokalizację pod postacią woluminu.

```
-v {PATH_TO_CONFIGS_DIR}:/app/config/
```

Można także zmienić lokalizację folderu z konfiguracją ustawiając zmienną środowiskową `CONFIG`. Może być ona względna do ścieżki `/app`.

Nie można zapomnieć o wyjątku konfiguracji `NLoga`, którą trzeba przekazać do folderu `/app`.

### 2.9.12. Przykład minimalnego systemu

Uruchamiając kontenery z modułami należy podać odpowiednie parametry aby aplikacje wewnątrz pracowały prawidłowo. Przykładowy minimalny system wraz z parametrami można zobaczyć w module `demonstration`[2]. Znajduje się tam system składający się z jednego brokera wiadomości (zewnętrzna i wewnętrzna komunikacja w jednym), panelu administracyjnego, jednego nadzorczy i jednego serwera wirtualizacji. Można znaleźć tam przykłady zarówno konfiguracji modułów jak i parametrów startowych dla kontenerów.

## 2.10. Interakcja z systemem

### 2.10.1. Typowe działanie systemu

Po uruchomieniu opisanym w 2.9.8 system powinien być w pełni używalny. Typowym zachowaniem systemu będzie cykliczne wysyłanie próśb przez nadzorców do serwerów wirtualizacji. W logach serwerów wirtualizacji jak i nadzorców powinny cyklicznie pojawiać się informacje o prośbie lub odpowiedzi na zapytanie o model.

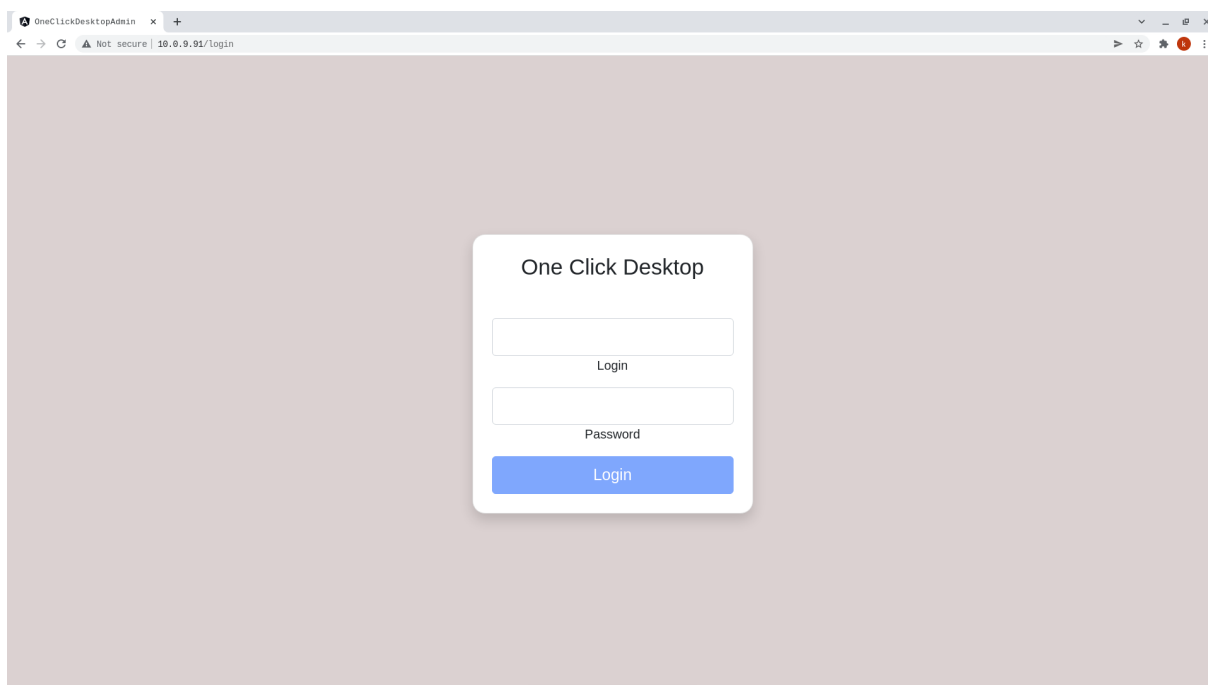
Zaraz po starcie systemu powinny pojawić się prośby o włączenie maszyn wirtualnych na odpowiednich serwerach wirtualizacji. Przy patrzeniu na standardowe wyjście aplikacji powinny być widoczne informacje od uruchamiającego się `Vagranta`. Przy każdej nowej utworzonej sesji powinny wybudzać się następne maszyny wirtualne. Często problemy z uruchomieniem maszyn wirtualnych będą oznaczać nieskończone próby uruchamiania maszyn wirtualnych, które zawsze będą kończyć się niepowodzeniem. Serwer wirtualizacji powinien wszystkie informacje o błędach zapisywać do pliku z logami we wnętrzu kontenera w folderze roboczym aplikacji `/app`.

Przy wyłączaniu serwerem wirtualizacji mogą pozostać nadal działające maszyny wirtualne wytworzone przez niego. Maszyny takie należy ręcznie wyłączyć lub usunąć. Nazwy maszyn można wyczytać z logów pracy serwera wirtualizacji.

## 2.10. INTERAKCJA Z SYSTEMEM

### 2.10.2. Funkcje panelu administracyjnego

Po wejściu na stronę panelu administracyjnego administrator musi podać nazwę użytkownika i hasło. Jeżeli rzeczywiście podane konto jest kontem administratora przejdzie dalej do panelu. W przeciwnym wypadku zostanie zwrócony błąd dostępu.



Rysunek 2.15: Okno logowania panelu administracyjnego

Po zalogowaniu administrator ma dostęp do głównego panelu z podsumowaniem wszystkich dostępnych zasobów. Po lewej stronie ma on dostępna listę wszystkich aktywnych serwerów wirtualizacji w systemie (rysunek 2.16).



Rysunek 2.16: Widok wszystkich zasobów systemu

Po naciśnięciu na wybrany serwer wirtualizacji na środku pojawiają się szczegóły zasobów konkretnego serwera (rysunek 2.17). Administrator Oprócz zasobów może sprawdzić ile aktualnie jest uruchomionych maszyn (sekcja Running) oraz ile maszyn można jeszcze uruchomić (sekcja Free).



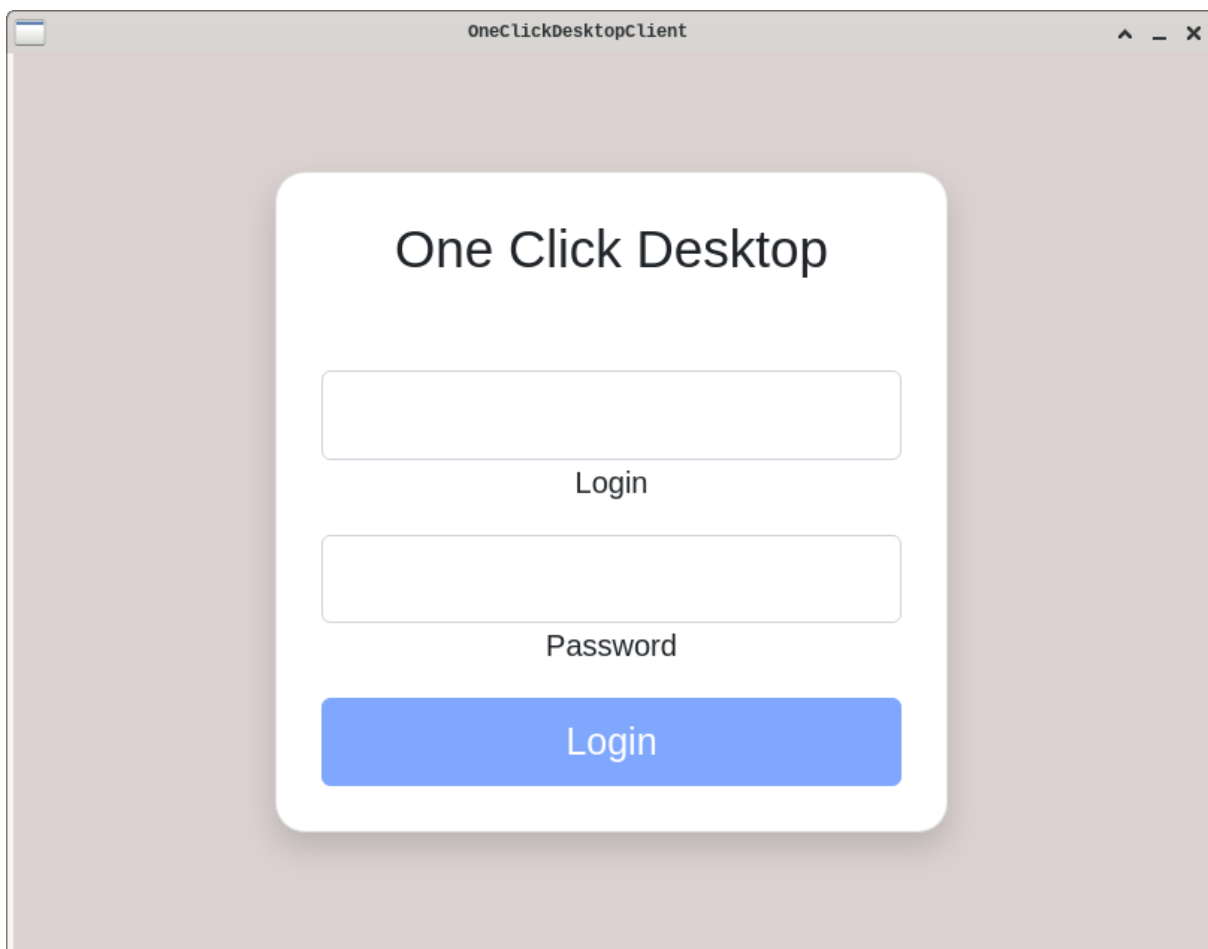
Rysunek 2.17: Widok szczegółowych zasobów serwera wirtualizacji

## 2.10. INTERAKCJA Z SYSTEMEM

Zawartość panelu odświeża się automatycznie.

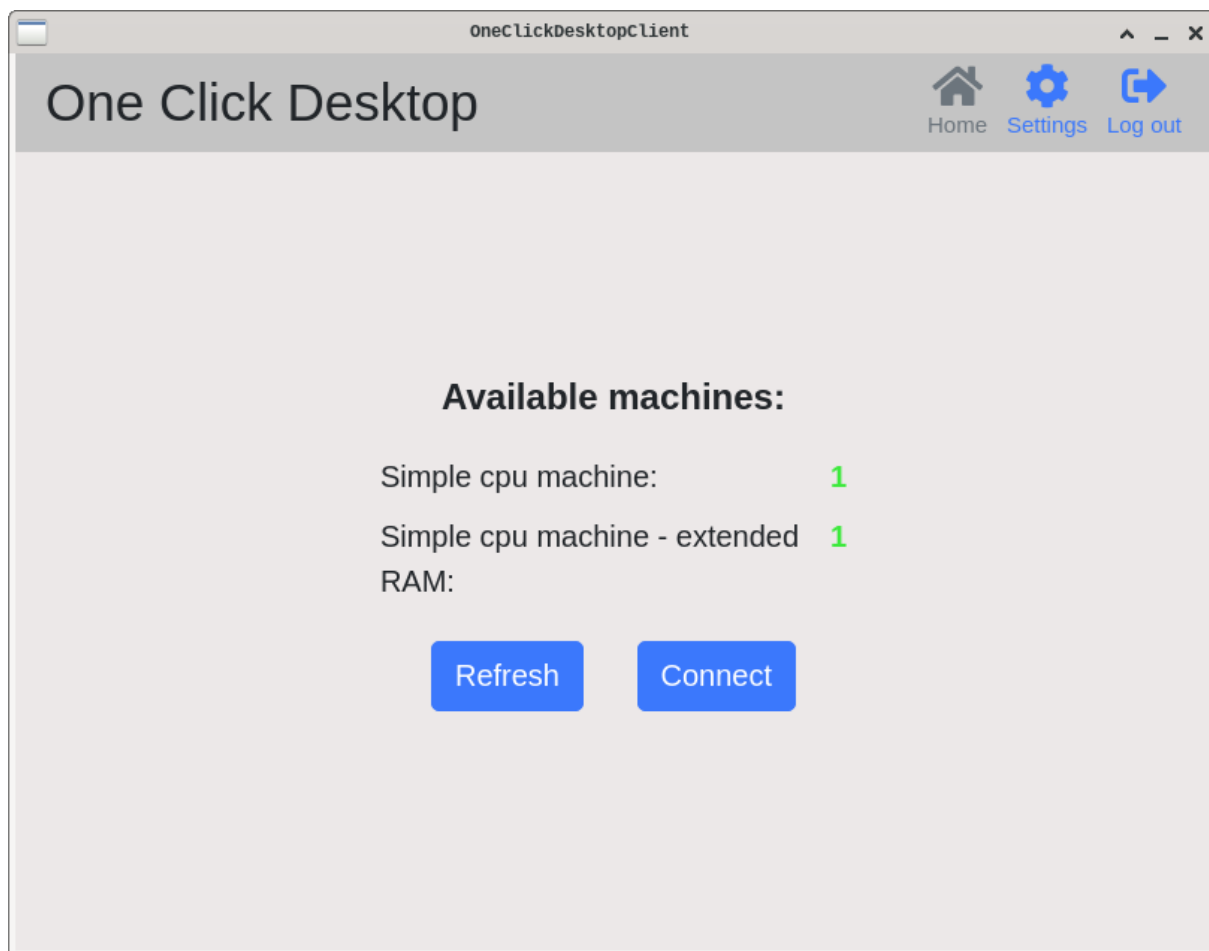
### 2.10.3. Funkcje aplikacji klienckiej

Po uruchomieniu aplikacji klienckiej użytkownik musi podać nazwę użytkownika i hasło. Jeżeli użytkownik istnieje w bazie to przejdzie do ekranu podsumowania. W przeciwnym wypadku zostanie zwrócony błąd dostępu.



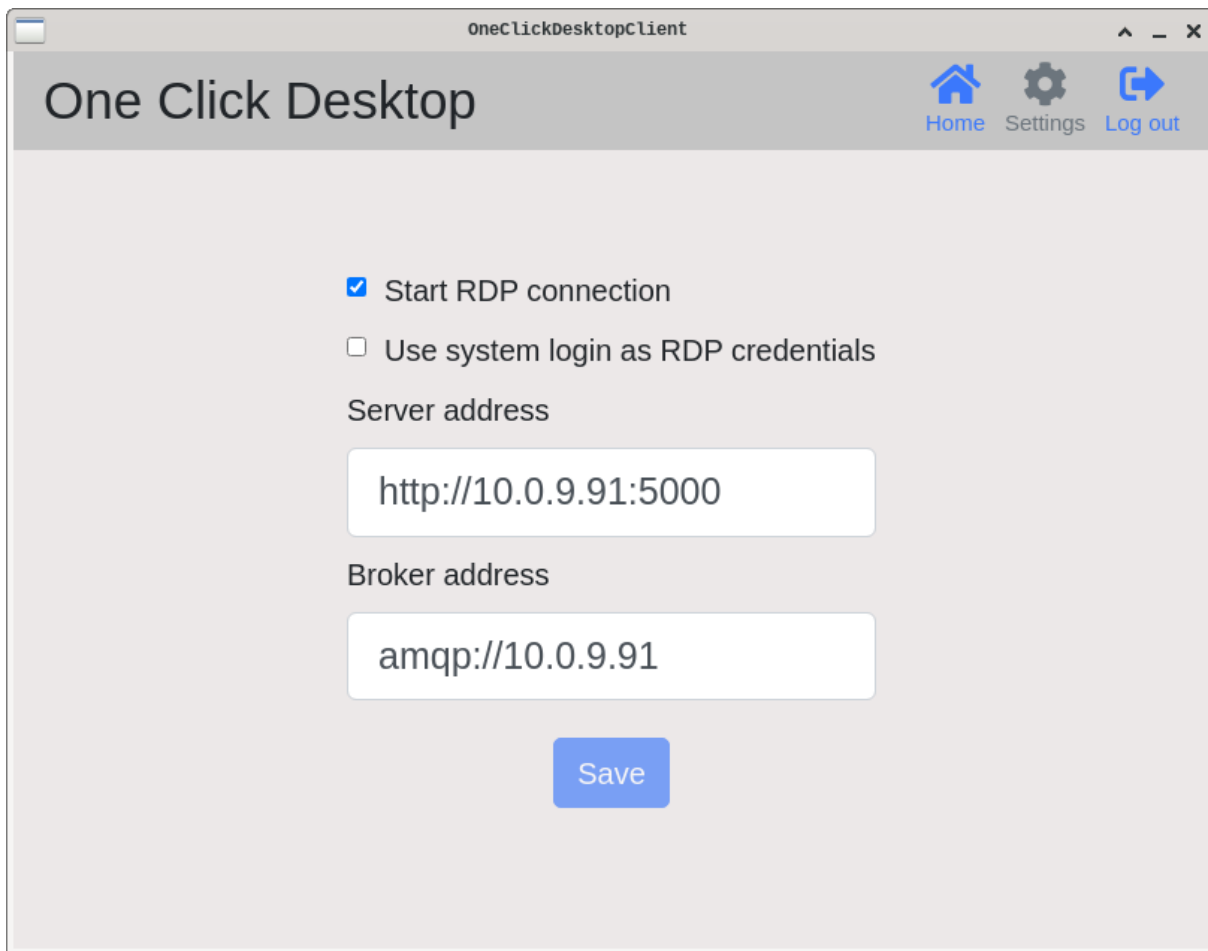
Rysunek 2.18: Ekran logowania aplikacji klienckiej

Po zalogowaniu użytkownik trafia do ekranu głównego (rysunek 2.19). Może tutaj zobaczyć ile jest dostępnych maszyn w systemie, poprosić o sesję oraz zmienić ustawienia aplikacji.



Rysunek 2.19: Ekran główny aplikacji klienckiej

Przechodząc do ustawień (rysunek 2.20) użytkownik może edytować plik konfiguracyjny z poziomu aplikacji albo wrócić do ekranu głównego. Zmiana adresu do nadzorca będzie wymagać ponownego uruchomienia. Inne ustawienia zostaną zastosowane od razu po zmianie.



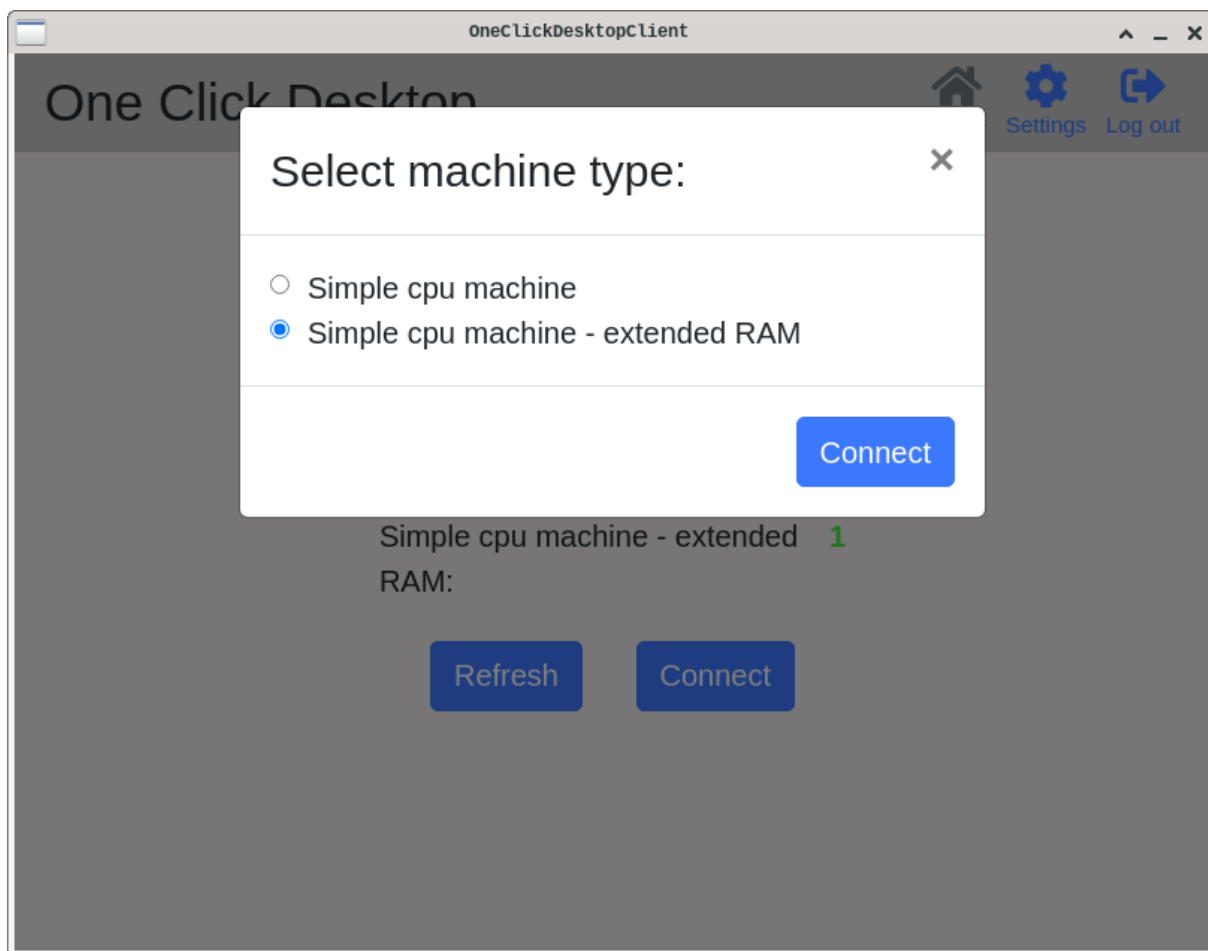
The screenshot shows a window titled "oneClickDesktopClient" with a header bar containing the text "One Click Desktop" and three navigation icons: "Home" (house icon), "Settings" (gear icon), and "Log out" (logout icon). The main content area has a light gray background and contains the following elements:

- A checked checkbox labeled "Start RDP connection".
- An unchecked checkbox labeled "Use system login as RDP credentials".
- A label "Server address" followed by a text input field containing "http://10.0.9.91:5000".
- A label "Broker address" followed by a text input field containing "amqp://10.0.9.91".
- A blue "Save" button centered below the input fields.

Rysunek 2.20: Ekran ustawień aplikacji klienckiej

Gdy użytkownik z ekranu głównego naciśnie przycisk *Connect* zostanie zapytany o typ sesji, jaki system am mu przypisać (rysunek 2.21).

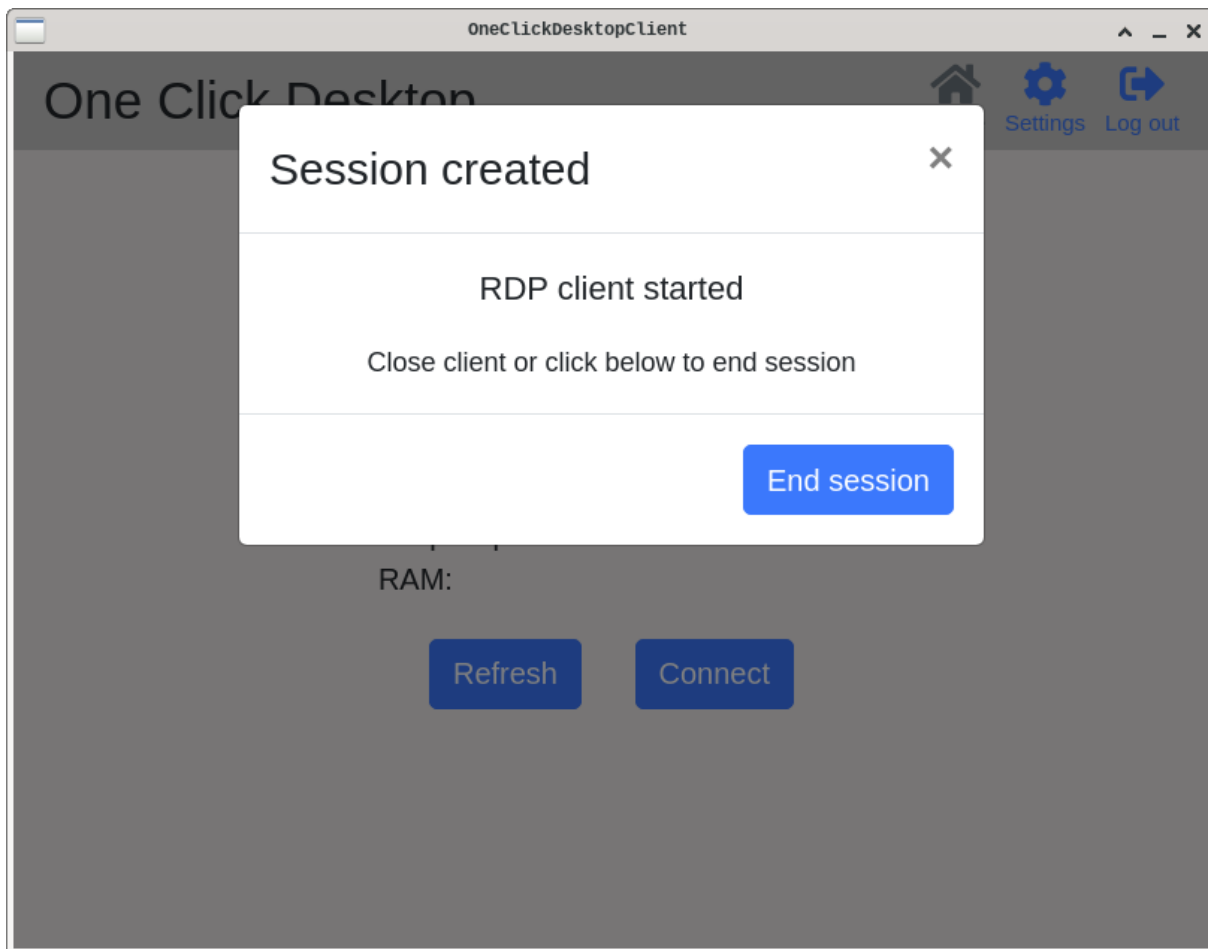




Rysunek 2.21: Ekran wyboru typu maszyny do pracy

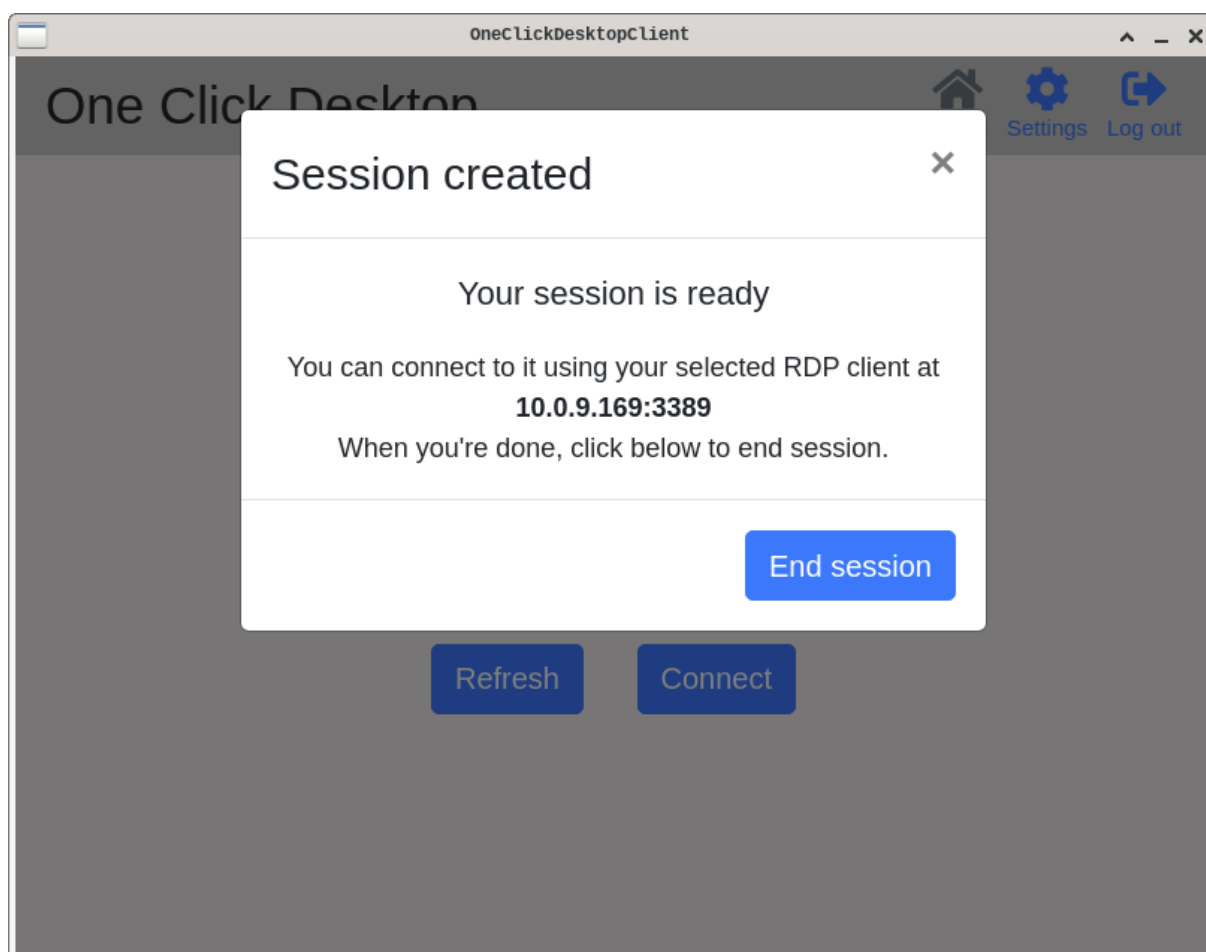
Po wybraniu jednego z typów sesji i naciśnięciu przycisku *Connect* aplikacja kliencka wyśle prośbę o znalezienie maszyny danego typu i utworzenie sesji dla pytającego użytkownika. Do czasu utworzenia sesji aplikacja będzie oczekiwać na dane do połączenia. W przypadku poprawnego utworzenia sesji uruchomi się klient RDP i będzie można rozpocząć pracę. W przeciwnym wypadku zostanie zgłoszony błąd i prośba o sesję zostanie umorzona.

Po prawidłowym utworzeniu sesji aplikacja rozpocznie zgłaszanie do zewnętrznego brokera wiadomości, że użytkownik pracuje na maszynie wirtualnej. Ten stan reprezentuje ekran pokazany na rysunku 2.22. Zamknięcie klienta RDP albo naciśnięcie przycisku *End session* spowoduje oznaczenie sesji jako *do usunięcia*. Od tego momentu użytkownik ma `DomainShutdownTimeout` minut na powrót do swojej sesji (szczegóły w 2.9.6). Po upływie tego czasu sesja zostaje zamknięta oraz maszyna z nią skojarzona wyłączona.



Rysunek 2.22: Ekran działającego połączenia RDP

W przypadku, gdy użytkownik poprosi aby nie uruchamiać dołączonego klienta RDP albo uruchomieni klienta RDP zakończy się błędem zostaną mu przekazane dane dostępowe do przypisanej maszyny wirtualnej (rysunek 2.23). Wtedy jedynym sposobem na zaznaczenie, że skończyło się prace na maszynie, jest naciśnięcie przycisku *End session*.



Rysunek 2.23: Ekran trwania połączenia w zewnętrznym kliencie RDP

## **3. Analiza rozwiązania**

### **3.1. Testowana funkcjonalność**

Gotowy system poddany został testom akceptacyjnym mającym za zadanie sprawdzić, czy działa on poprawnie, oraz czy spełnienia wymagania przedstawione w 1.5 oraz 1.6. W opisie testów, jako standardowe uruchomienie systemu rozumiemy procedurę opisaną w 2.9.8.

Wykonane zostały następujące scenariusze akceptacyjne:

#### **3.1.1. Brak komunikacji z nadzorcą**

Przy starcie samodzielnego serwera wirtualizacji i wykryciu braku komunikacji z jakimkolwiek nadzorcą (poprzez brokera wiadomości) serwer powinien poinformować o błędzie i zakończyć działanie.

Kroki:

1. Włącz wewnętrznego brokera wiadomości oraz serwer wirtualizacji.
2. Wyświetl błąd i zakończ działanie.

#### **3.1.2. Utrata komunikacji z nadzorcą**

Po poprawnym starcie systemu z pojedynczym nadzorcą oraz serwerem wirtualizacji, serwer powinien wyłączyć się po wyłączeniu się ostatniego (jedyne) nadzorcy.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Wyłącz nadzorcę lub brokery wiadomości.
4. Poczekaj na wykrycie braku nadzorców (brak otrzymanych wiadomości)
5. Wyświetl błąd i zakończ działanie.

#### 3.1.3. Standardowe użycie systemu przez użytkownika

Użytkownik podłącza się do systemu składającego się z pojedynczego nadzorcy oraz serwera wirtualizacji, gdzie działa przynajmniej jedna wolna maszyna. Użytkownik powinien prawidłowo otrzymać sesję, a po odłączeniu się od maszyny, powinna ona zostać wyłączona po 15 minutach (czas konfigurowalny).

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekać na uruchomienie przynajmniej jednej maszyny wirtualnej.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Zakończ sesję z poziomu aplikacji.
6. Po określonym czasie maszyna powinna się wyłączyć.

#### 3.1.4. Standardowe użycie systemu przez użytkownika przy awarii nadzorcy

Użytkownik podłącza się do systemu składającego się z dwóch nadzorców oraz jednego serwera wirtualizacji, gdzie działa przynajmniej jedna wolna maszyna. Użytkownik uzyskuje sesję, a w trakcie jej użytkowania następuje awaria nadzorcy. Po odłączeniu się od sesji i ponownej prośbie o sesję, w czasie krótszym niż czas wyłączenia maszyny, użytkownik powinien otrzymać ponownie tę samą maszynę.

Kroki:

1. Uruchom system standardową procedurą z dwoma nadzorcami.
2. Poczekać na uruchomienie przynajmniej jednej maszyny wirtualnej.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Wyłącz tego nadzorcę, z którym klient się komunikował.
6. Zakończ sesję z poziomu aplikacji.
7. Poproś ponownie o sesję poprzez aplikację kliencką (powinien uzyskać tę samą maszynę)
8. Podłącz się poprzez RDP do uzyskanej maszyny.

#### 3.1.5. Podłączenie nowego serwera wirtualizacji

W trakcie działania systemu nowy serwer wirtualizacji powinien zostać włączony do modelu nadzorców oraz wyświetlony w panelu administratora.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Włącz kolejną instancję serwera wirtualizacji.
4. Poczekaj na aktualizację modelu.
5. Sprawdź w panelu administratora, czy dwa serwery są w modelu.

#### 3.1.6. Podłączenie nowego nadzorcy

W trakcie działania systemu nowy nadzorca powinien posiadać taki sam model, jak aktualnie działający

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Włącz kolejną instancję nadzorcy.
4. Poczekaj na aktualizację modelu.
5. Sprawdź model na pierwszym nadzorcy poprzez panel administratora.
6. Sprawdź model na drugim nadzorcy poprzez panel administratora.

#### 3.1.7. Odnotowanie utraty serwera wirtualizacji

W trakcie działania systemu, przy utracie serwera wirtualizacji, nadzorcy powinni usunąć go z modelu.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Wyłącz serwer wirtualizacji.

### 3.1. TESTOWANA FUNKCJONALNOŚĆ

4. Poczekaj na odnotowanie straty.
5. Sprawdź model poprzez panel administratora.

#### 3.1.8. Utrata komunikacji przy działającej sesji

W trakcie działania systemu, przy utracie ostatniego nadzorcy, serwer wirtualizacji powinien zakończyć działanie. Jeżeli serwer posiada działające sesje, przed zakończeniem pracy musi poczekać na ich zakończenie.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.
5. Wyłącz nadzorcę lub brokery wiadomości.
6. Poczekaj na wykrycie braku nadzorców (brak otrzymanych wiadomości)
7. Wyświetl błąd, ale kontynuuj działanie.
8. Poczekaj na zakończenie sesji.
9. Zakończ działanie.

#### 3.1.9. Odzyskanie komunikacji przy działającej sesji

W trakcie działania systemu, przy utracie ostatniego nadzorcy, serwer wirtualizacji powinien zakończyć działanie. Jeżeli serwer posiada działające sesje, przed zakończeniem pracy musi poczekać na ich zakończenie. Jeżeli przed zakończeniem ostatniej sesji nastąpi przywrócenie komunikacji, serwer powinien kontynuować działanie po zakończeniu sesji.

Kroki:

1. Uruchom system standardową procedurą.
2. Poczekaj na prawidłowy start systemu.
3. Poproś o sesję poprzez aplikację kliencką.
4. Podłącz się poprzez RDP do uzyskanej maszyny.

5. Wyłącz nadzorcę lub brokery wiadomości.
6. Poczekaj na wykrycie braku nadzorców (brak otrzymanych wiadomości)
7. Wyświetl błąd, ale kontynuuj działanie.
8. Włącz wyłączony wcześniej moduł.
9. Poczekaj na zakończenie sesji.
10. Kontynuuj działanie.

### 3.2. Środowisko testowe

Testy przeprowadzaliśmy na dwóch komputerach podłączonych do wspólnej sieci lokalnej. Usługa OpenLDAP była uruchomiona na innym komputerze oraz udostępniała testową bazę użytkowników. Każdy z nich pracował pod kontrolą systemu operacyjnego Arch Linux w wersji ze stycznia 2022 roku. Obydwa pracowały z 8 GB pamięci operacyjnej oraz 4 rdzeniowym procesorem o 8 wątkach (Intel i7-2600K oraz i7-4790). Do testów skonfigurowaliśmy serwery wirtualizacji, aby miały do dyspozycji 6 wątków oraz 4096 MB pamięci operacyjnej.

Niestety taka platforma uniemożliwiła przetestowanie funkcjonalności PCI Passthrough (patrz wymagania sprzętowe *odnośnik!*). Budowa ich płyt głównych nie pozwalała na całkowitą izolację urządzeń podłączonych do złączy PCI Express. Skorzystaliśmy z innego komputera, który posiadał inną konstrukcję płyty głównej i każde z urządzeń zostało prawidłowo odizolowane dzięki przydzieleniu innych adresów pamięci. Pracował on pod kontrolą systemu operacyjnego Arch Linux ze stycznia 2022 roku. Posiadał on 48GB pamięci operacyjnej oraz 12 rdzeniowy procesor o 24 wątkach (AMD Threadripper 1920X). Uruchamiany na nim serwer wirtualizacji miał do dyspozycji 20 wątków oraz 43008 MB pamięci operacyjnej.

Na każdym z tych komputerów można było uruchomić aplikację serwera wirtualizacji, panelu administracyjnego oraz nadzorcy.

### 3.3. Wykonane testy

W trakcie rozwoju projektu zaprogramowaliśmy wiele testów automatycznych, które sprawdzały podstawową logikę wydzielonych części kodu. W przypadku modułów korzystających z zewnętrznych narzędzi (np. libvirt) stworzyliśmy proste testy integracyjne. Sprawdzają czy wszystkie wykorzystywane funkcjonalności tych narzędzi zostały prawidłowo zintegrowane.



### 3.4. WYNIKI TESTÓW

Po zakończeniu rozwoju funkcjonalności systemu przystąpiliśmy do próby wdrożenia systemu w środowiskach testowych. Wykonaliśmy wtedy ręcznie podstawowe scenariusze uruchomienia systemu w postaci kontenerów dockerowych. Gdy już system pracował stabilnie w takiej postaci przystąpiliśmy do ręcznego wykonywania scenariuszy akceptacyjnych

#### 3.4. Wyniki testów

Testy automatyczne często kończyły się błędem po zmianie w kodzie. Był dla nas znak aby sprawdzić co się dzieje i wprowadzić odpowiednie poprawki.

Próby wdrożenia systemu w środowisku testowych przyniosły odkrycie wielu błędów logicznych w naszych wstępnych pomysłach jak i samej implementacji. Dodatkowo pomogło nam to też lepiej zrozumieć wymagania sprzętowe jak i zależności naszego systemu.

Przy wykonywaniu testów akceptacyjnych mieliśmy szansę dopracować stabilność oraz ponownie skonfrontować wstępne pomysły z rzeczywistością. Po poprawkach błędów i upewnieniu się, że wszystkie scenariusze akceptacyjne są już spełnione system był już wystarczająco stabilny aby można było z niego korzystać. Jednak zrozumieliśmy wtedy lepiej jakie rozwiązania nie są wygodne i wymagają poprawek albo nawet przeprojektowania. Nasze wnioski o rozwiązaniach do poprawki można znaleźć w podsumowaniu (4.1).

## 4. Podsumowanie

### 4.1. Końcowy stan systemu

System, w swojej ostatecznej postaci, spełnia wszystkie postawione mu wymagania. Mimo trudności w implementacji niektórych funkcjonalności, w szczególności zarządzania maszynami wirtualnymi z poziomu kodu `C#`, ostateczny projekt działa zgodnie z oczekiwaniami. Projekt tworzony w ramach tej pracy uznajemy zatem za udany.

Elementem, który mógł zostać lepiej wykonany jest zarządzanie maszynami wirtualnymi przy pomocy Vagranta. Z powodu braku interfejsu programistycznego udostępnianego przez ten program, wykorzystywany jest on w naszym systemie za pośrednictwem wywołań poleceń powłoki. Nie jest to najlepsze rozwiązanie, o czym świadczy liczba błędów, które wynikły podczas tworzenia systemu. Jest to w naszym przekonaniu najbardziej podatny na błędy element systemu. Taki sposób rozwiązania jest spowodowany chęcią skorzystania z Vagranta podczas projektowania systemu. Na tamtym etapie nie sprawdziliśmy, czy dostępne są narzędzia umożliwiające korzystanie z niego z poziomu kodu. W momencie odkrycia braku takich narzędzi, było już za późno na zastąpienie Vagranta innym rozwiązaniem.

Projektując komunikację wewnętrzną za pośrednictwem RabbitMQ postawiliśmy na wykorzystanie po jednej kolejce na odbieranie wiadomości bezpośrednich oraz zbiorczych. W połączeniu z faktem, że przesyłane wiadomości posiadają różną formę, doprowadziło to do potrzeby deserializacji wiadomości na różne typy obiektów, w zależności od typu otrzymanej wiadomości. Naszym zdaniem problem ten nie został przez nas rozwiązany zadowalająco i mógłby zostać wykonany lepiej.

Z pominięciem wyżej wymienionych aspektów, uważamy projekt komunikacji i procesów biznesowych za dobrze wykonany, w sposób umożliwiający łatwą implementację. Dzięki głęboko przemyślanym rozwiązaniom w tych płaszczyznach, ich implementacja była mało problematyczna. Podczas testów nie wynikły również żadne problemy, które wymagałyby modyfikacji ustalonych procesów i sekwencji.

### 4.2. Możliwe ścieżki rozwoju

#### 4.2.1. Panel administratora

Funkcjonalność panelu administratora jest obecnie ograniczona. Pozwala on jedynie na podejście do stanu zasobów w systemie. Informacja ta pozwala ocenić, czy potrzebne jest uruchomienie nowych instancji serwera wirtualizacji. Jest to jednak jedyna dostępna informacja.

Możliwym rozwinięciem jest udostępnienie wglądu w cały model systemu przechowywany przez nadzorców. Umożliwi to ocenę stopnia zajętości maszyn każdego typu oraz ilości użytkowników systemu. Dodatkowo wgląd do stanu konkretnych maszyn i sesji może ułatwić rozwiązywanie problemów.

#### 4.2.2. Użycie konkretnych kart graficznych

W konfiguracji typu maszyny wirtualnej możliwe jest jedynie wskazanie, czy ma ona posiadać kartę graficzną na wyłączność. Oznacza to, że otrzymana karta graficzna nie jest sprecyzowana i może być dowolną ze skonfigurowanych w serwerze wirtualizacji.

W celu uniknięcia niespodzianek, co do modelu otrzymanej karty graficznej, preferowane byłaby możliwość sprecyzowania konkretnego modelu karty graficznej przekazywanej do maszyny danego typu. Możliwym sposobem osiągnięcia tej funkcjonalności jest rozszerzenie konfiguracji serwera wirtualizacji o możliwość nadania identyfikatora przekazywanym kartom graficznym.

### 4.3. Otwarte problemy

#### 4.3.1. Konfiguracja połączenia RabbitMQ

Klient RabbitMQ używany przez moduły wewnętrzne do połączenia z instancją brokera udostępnia wiele opcji konfiguracji, które nie są udostępniane przez moduły. Konfiguracja ta definiuje opcje połączenia, takie jak hasło autoryzacji, czy niestandardowe ścieżki połączenia. W przypadku korzystania z niestandardowej instancji brokera, konfiguracja ta może okazać się niezbędna. Z tego powodu powinno być możliwe przekazanie ustawień połączenia z brokerem wiadomości w plikach konfiguracyjnych modułów wewnętrznych.

#### 4.3.2. Monitorowanie stanu połączenia klienta

Do monitorowania stanu połączenia z maszyną wirtualną wykorzystywany jest broker wiadomości RabbitMQ. Klient uznawany jest za połączony gdy istnieje kolejka z nazwą odpowia-

dającą identyfikatorowi sesji. Rozwiązanie to jest bardzo niestandardowe, kolejki używane są w nie zamierzony sposób. Dodatkowo klient RabbitMQ nie udostępnia możliwości powiadamiania o braku kolejki w sposób pasywny. Implementacja ta wymaga działania instancji brokera, która jest dostępna spoza wnętrza systemu. Może to powodować luki w bezpieczeństwie. Pożądane byłoby zastąpienie tej implementacji mechanizmem przeznaczonym do tego typu rozwiązań.

#### 4.3.3. Znane problemy wyścigów

Pomimo przywiązania dużej uwagi do zaprojektowania komunikacji i procesów w sposób niwelujący ilość możliwych problemów wynikających z konkurencji, wciąż istnieją miejsca, w których mogą one wystąpić. Znanym problemem jest możliwość uruchomienia dwa razy więcej maszyn niż zamierzono podczas startu systemu. Spowodowane jest to tym, że nadzorca otrzymując informację o dostępności serwera wirtualizacji poprosi go o utworzenie wymaganych maszyn wirtualnych. Jeżeli zanim serwer prześle informację o wykonaniu zadania, drugi serwer wirtualizacji również stanie się dostępny, nadzorca poprosi go o utworzenie tych samych maszyn. Taki ciąg wydarzeń doprowadzi do działania dwa razy większej liczby maszyn niż było zamierzone.

Możliwym rozwiązaniem problemu może być przeprojektowanie procesu uruchamiania wymaganych maszyn, w celu uwzględnienia możliwości pojawienia się kolejnych instancji serwera wirtualizacji.

## Bibliografia

### Repozytoria kodu

- [1] P.Widomski K.Smogór. *Przykładowy VagrantBox smogork/archlinux-rdp*. <https://app.vagrantup.com/smogork/boxes/archlinux-rdp>. [Online; dostęp 21.01.2021r.]
- [2] P.Widomski K.Smogór. *Repozytorium z demonstracyjnym systemem*. <https://github.com/one-click-desktop/demonstration>. [Online; dostęp 21.01.2021r.]
- [3] P.Widomski K.Smogór. *Repozytorium z konfiguracją systemu*. <https://github.com/one-click-desktop/configuration>. [Online; dostęp 21.01.2021r.] 2022.
- [4] P.Widomski K.Smogór. *Strona z wydaniem aplikacji klienckiej*. <https://github.com/one-click-desktop/client/releases>. [Online; dostęp 21.01.2021r.] 2022.

### Źródła zewnętrzne

- [5] *Lista klientów RDP zgodnych z serwerem XRDP*. <https://github.com/neutrinolabs/xrdp#overview>. [Online; dostęp 21.01.2021r.] 2021.
- [6] *Mechanizm odrzucania niedostarczonych wiadomości w RabbitMQ*. <https://www.rabbitmq.com/publishers.html#unroutable>. [Online; dostęp 21.01.2021r.]
- [7] *Mechanizm potwierdzenia otrzymania wiadomości w RabbitMQ*. <https://www.rabbitmq.com/confirmations.html>. [Online; dostęp 21.01.2021r.]
- [8] *Oferta Citrixa usług zdalnego pulpitu*. <https://www.citrix.com/pl-pl/products/citrix-virtual-apps-and-desktops/>. [Online; dostęp 21.01.2021r.]
- [9] *Rozdział dokumentacji Vagranta o tworzeniu boxów*. <https://www.vagrantup.com/docs/boxes/base>. [Online; dostęp 21.01.2021r.]
- [10] *Rozdział dokumentacji Ansible'a o playbookach*. [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html). [Online; dostęp 21.01.2021r.] 31.03.2021r.

- [11] *Rozdział dokumentacji libvirta o adresie dostępu.* <https://libvirt.org/uri.html>. [Online; dostęp 21.01.2021r.]
- [12] *Rozdział dokumentacji vagrant-libvirt o tworzeniu boxów.* <https://github.com/vagrant-libvirt/vagrant-libvirt#create-box>. [Online; dostęp 21.01.2021r.]
- [13] *Rozdział dokumentacji Vagranta o boxach.* <https://www.vagrantup.com/docs/boxes>. [Online; dostęp 21.01.2021r.] 31.03.2021r.
- [14] *Rozdział w dokumentacji Ansible o wymaganiach komunikacji.* <https://docs.ansible.com/ansible/latest/plugins/connection.html#connection-plugins>. [Online; dostęp 21.01.2021r.]
- [15] *Understanding the Remote Desktop Protocol (RDP).* <https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol>. [Online; dostęp 21.01.2021r.] 24.09.2021r.

## Wykaz symboli i skrótów

RDP Remote Desktop Protocol

\* operator gwiazdka

~ tylda

## Spis rysunków

1.1	Przypadki użycia aplikacji nadzorczej . . . . .	15
1.2	Przypadki użycia serwera wirtualizacji . . . . .	17
1.3	Przypadki użycia panelu administratora . . . . .	19
2.1	Schematyczna architektura systemu . . . . .	27
2.2	Schemat klas modelu systemu . . . . .	28
2.3	Diagram stanów dla maszyny wirtualnej . . . . .	32
2.4	Diagram stanów dla serwera wirtualizacji . . . . .	33
2.5	Diagram stanów dla użytkownika . . . . .	34
2.6	Diagram aktywności dla uzyskania sesji . . . . .	35
2.7	Diagram aktywności dla zakończenia sesji . . . . .	36
2.8	Diagram aktywności dla rozpoczęcia pracy serwera wirtualizacji . . . . .	37
2.9	Endpointy API . . . . .	39
2.10	Sekwencja komunikacji utworzenia sesji . . . . .	41
2.11	Sekwencja komunikacji zakończenia sesji . . . . .	42
2.12	Sekwencja komunikacji aktualizacji stanu systemu . . . . .	42
2.13	Sekwencja komunikacji włączenia maszyny . . . . .	43
2.14	Sekwencja komunikacji wyłączenia maszyny . . . . .	43
2.15	Okno logowania panelu administracyjnego . . . . .	63
2.16	Widok wszystkich zasobów systemu . . . . .	64
2.17	Widok szczegółowych zasobów serwera wirtualizacji . . . . .	64
2.18	Ekran logowania aplikacji klienckiej . . . . .	66
2.19	Ekran główny aplikacji klienckiej . . . . .	67
2.20	Ekran ustawień aplikacji klienckiej . . . . .	68
2.21	Ekran wyboru typu maszyny do pracy . . . . .	69
2.22	Ekran działającego połączenia RDP . . . . .	70
2.23	Ekran trwania połączenia w zewnętrznym kliencie RDP . . . . .	71



## Spis tabel

1.1	Przypadki użycia aplikacji nadzorczej . . . . .	16
1.2	Przypadki użycia serwera wirtualizacji . . . . .	18
1.3	Przypadki użycia panelu administratora . . . . .	19
1.4	Opis skrócony . . . . .	21
1.5	Analiza ryzyka . . . . .	22

## Spis załączników

1. Załącznik 1
2. Załącznik 2
3. Jak nie występują, usunąć rozdział.