

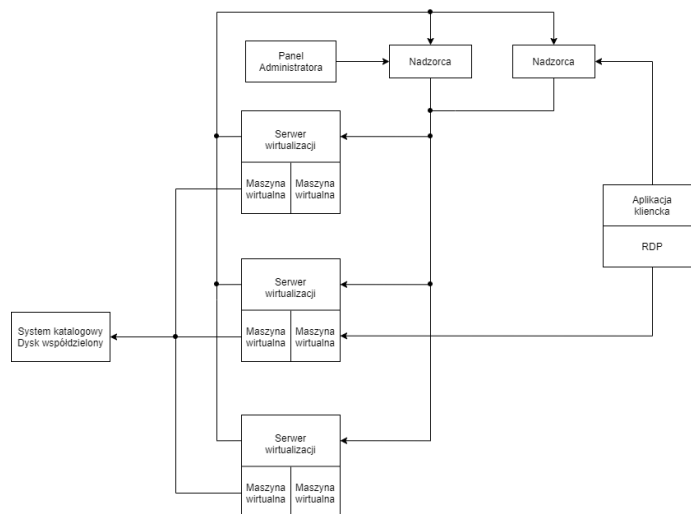
# 1 BLa bla bla - wymagane punkty jak z deliverable-one (MODULARNOSC!!!)

## 2 Architektura systemu

Przedstawiony system składa się z następujących modułów:

- nadzorcy,
- serwer virtualizacji,
- brokera wiadomości,
- aplikacji klienckiej,
- panelu administratora,
- systemu katalogowego,
- dysku współdzielonego.

Schematyczny obraz systemu przedstawia poniższy rysunek.



Rysunek 1: Schematyczna architektura systemu

Z założenia system powinien móc skalować się w dwóch wymiarach, to znaczy:

1. Zwiększanie liczby serwerów wirtualnych - zwiększenie liczby sesji dla użytkowników.
2. Zwiększenie liczby nadzorców - zwiększenie liczby obsługiwanych klientów jednocześnie.

Szczegółowe opisy poszczególnych modułów będą omówione w następnych rozdziałach: Opis tworzonych modułów oraz Opis zewnętrznie dostarczonych modułów.

## 3 Opis tworzonych modułów

3.1 Nadzorca

3.2 Serwer wirtualizacji

3.3 Aplikacja kliencka

3.4 Panel administratora

## 4 Opis zewnętrznie dostarczonych modułów

### 4.1 Broker wiadomości

Komunikacje wewnątrz systemu, czyli pomiędzy serwerami wirtualizacji oraz nadzorcami, będzie realizowali poprzez kolejki wiadomości. Wiadomości będą przekazywane pomiędzy modułami przez brokera, którego nie będziemy implementować. Zdecydowaliśmy się na skorzystanie z systemu RabbitMQ, który zapewni niezawodność komunikacji pomiędzy modułami. Pozostaje jednak problem hazardu oraz wyścigów, które zostaną wyeliminowane w logice komunikacji nadzorcy oraz serwera wirtualizacji.

Aby system mógł funkcjonować zdefiniowane zostaną kolejki:

- (I) Kolejka kończąca się na każdym z serwerów wirtualizacji i dla każdego z nich wiadomości są powielane. Służy ona do wysyłania próśb od nadzorców do serwerów wirtualizacji.
- (II) Kolejka kończąca się na każdym z nadzorców i dla każdego z nich wiadomości są powielane. Służy ona do wysyłania informacji do nadzorców o zmianie stanu w serwerze wirtualizacji. Dodatkowo może służyć do wymiany danych pomiędzy nadzorcami.
- (III) Kolejka kończąca się wyłącznie na pojedynczym serwerze wirtualizacji. Liczba kolejek zgadza się z liczbą serwerów wirtualizacji aktywnych w systemie. Służą one do sprawdzania, czy serwer wirtualizacji nadal pracuje po drugiej stronie. Skorzystamy z funkcjonalności kolejek na wyłączność (Exclusive Queue<sup>1</sup>). Każda z nich powinna mieć nazwę jaka przedstawi się serwer wirtualizacji.

Powyższe 3 grupy kolejek umożliwią prawidłowe działanie systemu. Każdy z modułów utworzy odpowiednie kolejki w trakcie uruchamiania. Jedynym wymogiem prawidłowego uruchomienia komunikacji jest dostępny przez wszystkie serwery wirtualizacji oraz nadzorców proces brokera.

### 4.2 Dysk sieciowy

Usługa wspólnej przestrzeni dyskowej jest potrzebna do uzyskania niezależności wyboru maszyny wirtualnej od folderu użytkownika. Każda maszyna wirtualna powinna przy starcie otrzymać adres oraz dane dostępowe do takiego dysku sieciowego. Dane zostaną dostarczone poprzez wykonanie Ansible playbooka po uruchomieniu maszyny.

W przypadku maszyn wirtualnych uruchamiających system Linux potrzebne są dane do połączenia przez protokół NFS, a przy systemie Windows dane do protokołu SAMBA. Niezależnie od protokołu dane nie mogą się różnić (struktura katalogów oraz zawartość plików).

---

<sup>1</sup>Opis zachowania kolejek na wyłączność

### 4.3 System katalogowy

Usługa systemu katalogowego jest potrzebna do uzyskania niezależności wyboru maszyny wirtualnej od danych logowania do systemu uruchomionego na maszynie wirtualnej. Każda maszyna wirtualna powinna przy starcie otrzymać adres oraz dane dostępne do takiego systemu katalogowego. Dane zostaną dostarczone poprzez wykonanie Ansible playbooka po uruchomieniu maszyny.

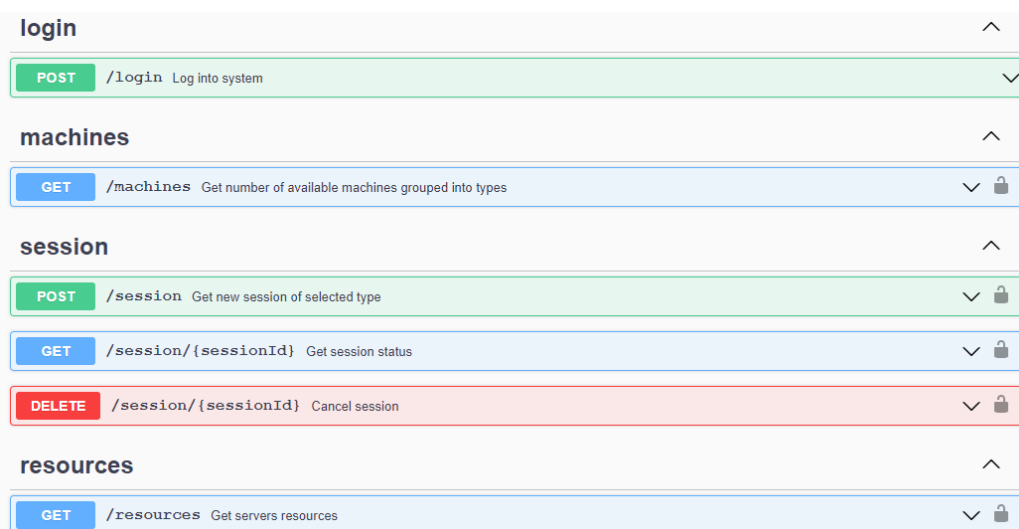
Aby system katalogowy był kompatybilny z systemami Windows oraz Linux uruchamianymi na maszynie wirtualnej skorzystamy z protokołu OpenLDAP do uzyskiwania danych o użytkownikach.

## 5 Komunikacja

### 5.1 Komunikacja użytkownika z systemem - REST API

Komunikacja aplikacji klienckiej oraz panelu administratora z systemem - nadzorcą - rozwiązana jest za pomocą REST API<sup>2</sup>. Wiadomości wysyłane są za pomocą protokołu HTTPS<sup>3</sup>, który zapewnia ich szyfrowanie. W tym celu wymagane jest, aby na adres, pod którym udostępniony będzie system, wystawiony był odpowiedni certyfikat<sup>4</sup>, gwarantujący jego tożsamość. Podczas tworzenia systemu i testów możliwe jest użycie sztucznego, własnoręcznie podpisanego certyfikatu<sup>5</sup>.

Całość specyfikacji API umieszczona jest w osobnym pliku. 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: 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. 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. Ten endpoint, oraz wszystkie następne wymagają autoryzacji poprzez umieszczenie tokenu otrzymanego podczas logowania w odpowiednim nagłówku wiadomości, oraz dostępne są tylko dla użytkownika.

<sup>2</sup>Opis REST API

<sup>3</sup>Specyfikacja protokołu HTTP Over TLS

<sup>4</sup>Opis certyfikatu TLS/SSL

<sup>5</sup>Opis własnoręcznie podpisanego certyfikatu TLS/SSL

- Session - pozwala na wysłanie prośby o uzyskanie sesji, pobranie stanu sesji oraz jej anulowanie.
- Resources - udostępnia informację o zasobach działających serwerów wirtualizacji. Dostępny jedynie dla administratora.

## 5.2 Komunikacja wewnątrz systemu - broker wiadomości

Do komunikacji wewnątrz systemowej użyty jest broker wiadomości RabbitMQ. Umożliwia on wysyłanie asynchronicznych wiadomości do wielu odbiorców jednocześnie. Dzięki temu żaden z modułów nie musi znać dokładnej liczby oraz adresów modułów, z którymi chce się komunikować. Wewnątrz systemu działa dwóch brokerów wiadomości. Jeden, główny, służy do komunikacji pomiędzy nadzorcami i serwerami wirtualizacji. Obsługuje on kanał, którego nadawcami są nadzorcy, a odbiorcami serwery wirtualizacji oraz kanał z odwrotną komunikacją. Wiadomości wysyłane przez nadzorcę do wszystkich serwerów wysyłane są pierwszą z kolejek,

## 6 Diagramy

6.1 Diagramy stanów

6.2 Diagramy aktywności

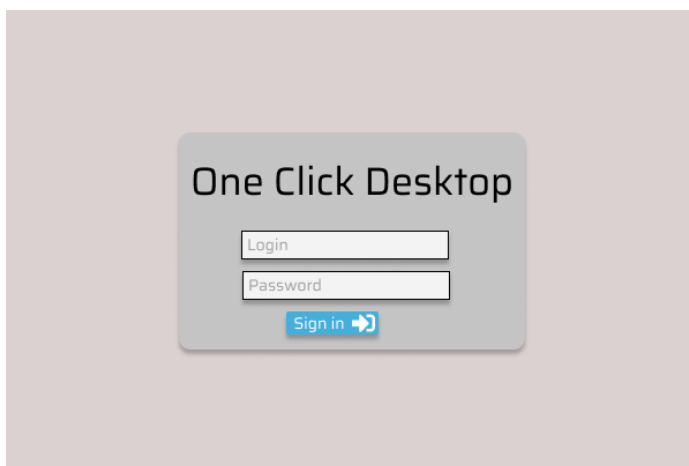
6.3 Diagramy klas

6.4 Diagramy sekwencji

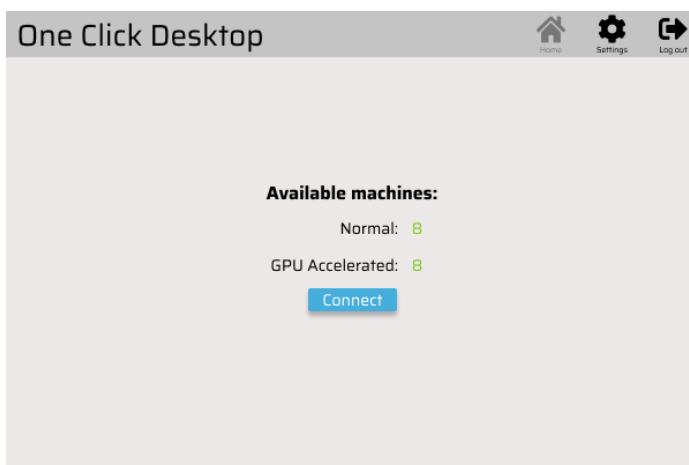
## 7 Interfejs użytkownika

### 7.1 Aplikacja kliencka

Aplikacja kliencka posiada interfejs użytkownika pozwalający na zalogowanie się oraz nawiązanie połączenia ze zdalną sesją. Użytkownikowi wyświetlany jest czynność, która aktualnie się odbywa, oraz w każdym momencie może zakończyć sesję.



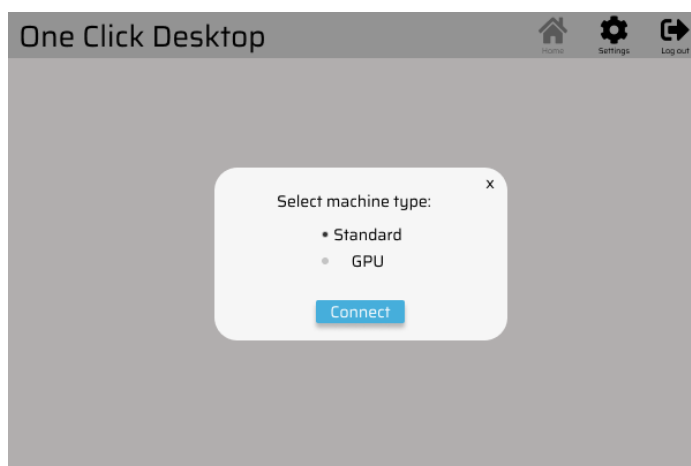
Rysunek 3: Ekran logowania



Rysunek 4: Główny widok zawierający dostępność maszyn

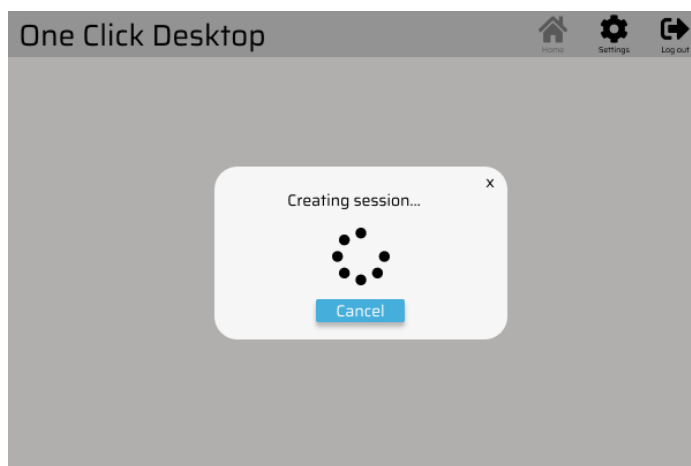
Na tym ekranie użytkownik może rozpocząć proces uzyskiwania sesji, przejść do ekranu ustawień lub wylogować się. Jeżeli w systemie nie ma dostępnych maszyn, lub nie uda się uzyskać informacji o ich dostępności, to przycisk połączenia jest niedostępny. Wciśnięcie tego przycisku prowadzi do kolejnego ekranu.



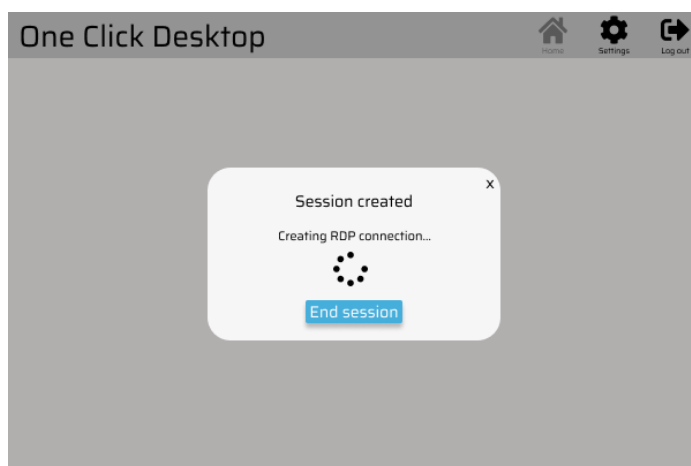


Rysunek 5: Wybór typu sesji

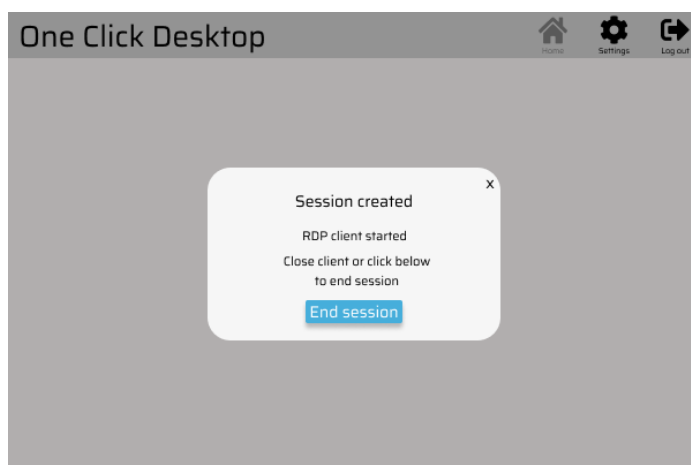
Do wyboru dostępne są jedynie typy sesji, które system określi jako dostępne. Wybranie typu sesji kliknięcie przycisku prowadzi do kolejnego ekranu. Następne ekrany przechodzą automatycznie do kolejnych bez interwencji użytkownika, aż do informacji o nawiązaniu połączenia lub błędu.



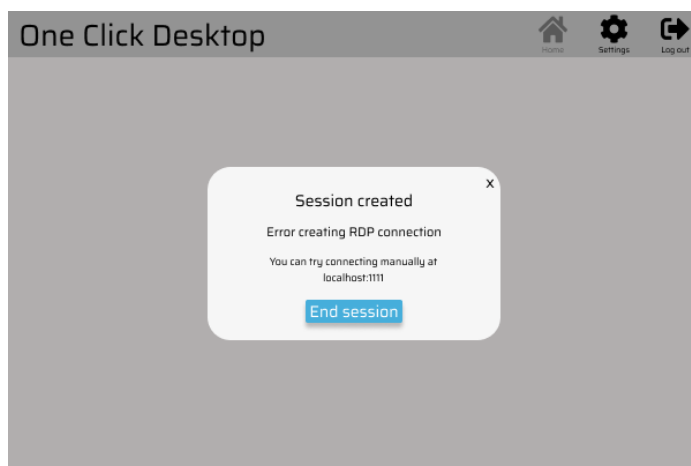
Rysunek 6: Tworzenie sesji



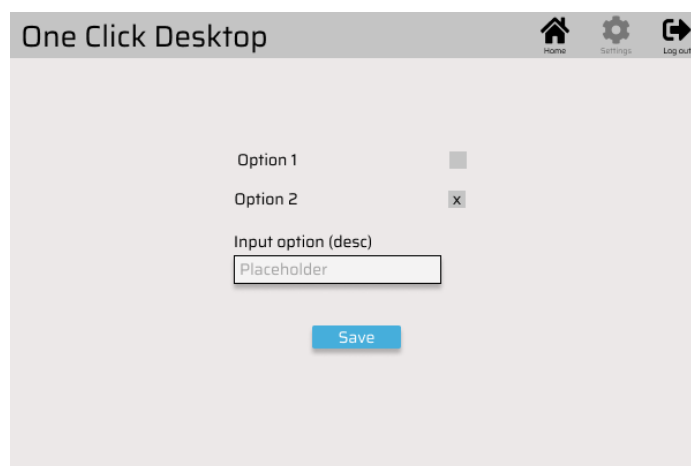
Rysunek 7: Nawiązywanie połączenia RDP



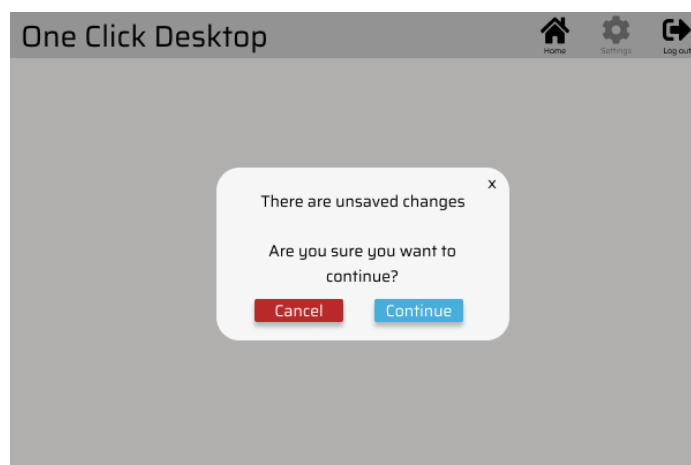
Rysunek 8: Połączenie nawiązane



Rysunek 9: Błąd przy nawiązywaniu połączenia



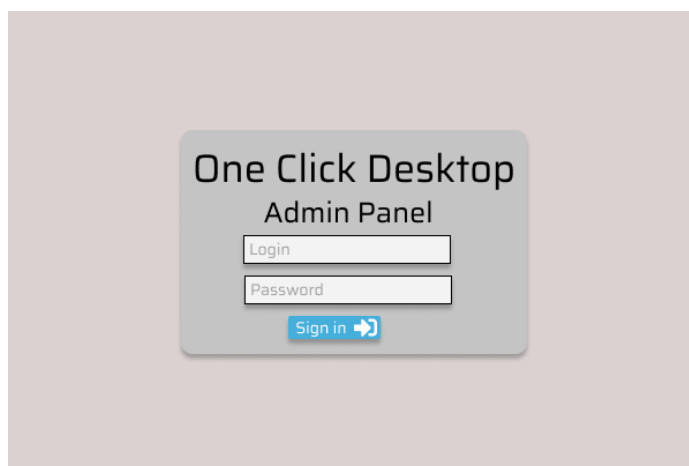
Rysunek 10: Ustawienia



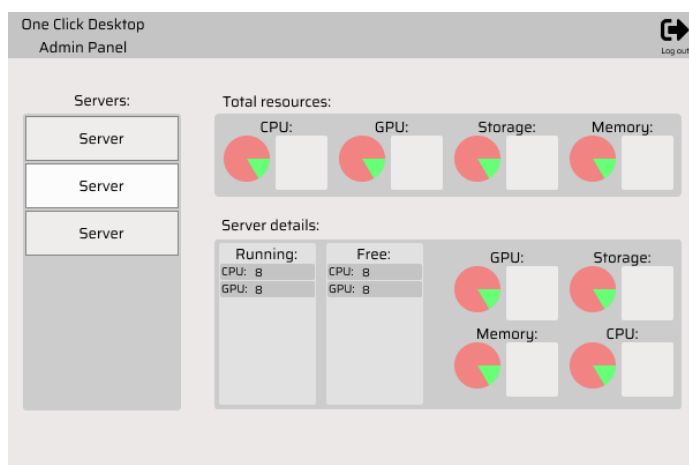
Rysunek 11: Powiadomienie przy wyjściu z ekranu ustawień z niezapisanymi zmianami

## 7.2 Panel administratora

Panel administratora posiada skromny interfejs umożliwiający zalogowanie się oraz podgląd zużycia zasobów.



Rysunek 12: Ekran logowania



Rysunek 13: Widok zużycia zasobów

Na tym widoku możemy zobaczyć zużycie zasobów globalne oraz dla każdego z serwerów wirtualizacji. Dodatkowo dla serwera wyświetlona jest również ilość działających i możliwych do uruchomienia maszyn każdego z typów.

## 8 Zewnętrzne narzędzia

### 8.1 Ansible

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

1. Dane dostępowe do dysku sieciowego oraz wykorzystany protokół
2. Dane dostępowe do usługi katalogowej
3. TODO: dopisać wszystkie potrzebne konfiguracje

Playbook można rozszerzać o potrzebne dane zależne od użycia.

### 8.2 Vagrant

Vagrant zostanie wykorzystany w celu łatwej parametryzacji oraz powtarzalnego tworzenia maszyn wirtualnych z przygotowanego wcześniej obrazu systemu. Głównie wykorzystany będzie mechanizm Vagrantboxów, 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. TODO: dopisać wszystkie potrzebne wymagania

### 8.3 Libvirt z QEMU

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

1. Tworzenie maszyn wirtualnych.
2. Uruchamianie maszyn wirtualnych.
3. Przyporządkowanie zasobów maszynom wirtualnym (w tym kraty graficzne).
4. Wyłączanie maszyn wirtualnych.
5. Sprawdzanie, czy maszyna działa na serwerze wirtualizacji.

## 9 Wybrana technologia

- Aplikacja kliencka
  - Typescript<sup>6</sup> /Javascript<sup>7</sup>
  - Node.js<sup>8</sup> - środowisko uruchomieniowe używane do integracji z systemem użytkownika
  - Angular<sup>9</sup> - renderowanie widoków
  - Electron<sup>10</sup> - platforma programistyczna
  - Jest<sup>11</sup> - testy jednostkowe
  - Cypress<sup>12</sup> - testy integracyjne
  - GNU/Linux oraz Windows - wspierane systemy operacyjne
- Panel administratora
  - Typescript/Javascript
  - Angular - platforma aplikacji WWW
  - Jest - testy jednostkowe
  - Cypress - testy integracyjne
- Nadzorca i serwer wirtualizacji
  - C#<sup>13</sup>
  - RabbitMQ<sup>14</sup> - broker asynchronicznych wiadomości
  - Ansible<sup>15</sup> - zarządzanie maszynami wirtualnymi
  - Vagrant<sup>16</sup> - tworzenie obrazów maszyn wirtualnych
  - libvirt<sup>17</sup> - uruchamianie maszyn wirtualnych
  - OpenLDAP<sup>18</sup> - dostępu do systemu katalogowego
  - NFS<sup>19</sup> - dostęp do katalogów domowych z maszyny wirtualnej

---

<sup>6</sup>Strona projektu Typescript

<sup>7</sup>Obecny standard języka Javascript

<sup>8</sup>Strona projektu Node.js

<sup>9</sup>Strona projektu Angular

<sup>10</sup>Strona projektu Electron

<sup>11</sup>Strona projektu Jest

<sup>12</sup>Strona projektu Cypress

<sup>13</sup>Dokumentacja języka C#

<sup>14</sup>Strona projektu RabbitMQ

<sup>15</sup>Strona projektu Ansible

<sup>16</sup>Strona projektu Vagrant

<sup>17</sup>Strona projektu libvirt

<sup>18</sup>Strona projektu libvirt

<sup>19</sup>Opis na stronie firmy Microfost

- Arch Linux<sup>20</sup> - system operacyjny uruchamiany przez maszyny wirtualne
- GNU/Linux - wspierany system operacyjny
- Różne
  - Swagger Codegen<sup>21</sup> - automatyczna generacja API na podstawie specyfikacji
  - RDP<sup>22</sup> - łączenie ze zdalnymi sesjami

---

<sup>20</sup>Strona systemu operacyjnego Arch Linux

<sup>21</sup>Opis narzędzia na stronie firmy Swagger

<sup>22</sup>Dokumentacja protokołu RDP od Microsoft