UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Plataforma online de pentesting

Autor: David Álvarez Ríos

Director: Raúl Roca
Realitzat a: Departamento de Matemátiques e Informática

Barcelona, 20 de junio de 2021

Abstract

The main objective for this work has been to offer to users access to a web page where they can solve cybersecurity problems in virtual machines without exiting the webpage.

The two major challenges I found when I started this project were how to visualize the virtual machines inside the web page and how to make the user interact with the virtual machines.

In this work we'll see how I faced this problems and how the have been resolved.

Resum

L'objectiu principal d'aquest treball ha sigut oferir a usuaris l'accés a una pàgina web on puguin resoldre problemes de ciberseguretat en màquines virtuals dins de la pròpia pàgina.

Els dos majors reptes que em vaig trobar al començar el projecte va ser com visualitzar les màquines virtuals dins de la pàgina i com poder fer que l'usuari l'utilitzi.

En aquest treball veurem com he afrontat els problemes presentats i com han sigut resolts.

Resumen

El objetivo principal de este trabajo ha sido el ofrecer a usuarios acceso a una página web donde puedan resolver problemas de ciberseguridad en máquinas virtuales dentro de la propia página.

Los dos mayores retos que me encontré nada más empezar este proyecto fueron en cómo visualizar las máquinas virtuales dentro de la página y cómo hacer que un usuario pueda utilizarla.

En este trabajo veremos cómo he afrontado los problemas presentados y cómo han sido resuelto.

Agradecimientos

“A mis padres y a toda mi familia, que han estado siempre apoyandome en mis estudios y siempre han estado en los buenos y malos momentos de la vida.”

“A los amigos que he hecho en la universidad, han sido un gran apoyo y he podido aprender mucho de todos ellos.”

“A mis mejores amigos desde el colegio, algunos los conozco desde hace más de 20 años, otros más de 10, pero siempre han estado ahí y solo tengo palabras de agradecimiento hacia ellos. Por muchas más décadas juntos.”

“A mi tutor de TFG, por su ayuda, guía y apoyo en este trabajo.”

Índice general

Introducción

Contexto

Uno de los problemas a la hora de aprender sobre ciberseguridad aparte de la complejidad de la misma, pues abarca un amplio campo de conocimientos, es el poder practicar los conceptos sin antes tener un amplio conocimiento de sistemas para preparar un entorno sobre el cual practicar. Este proyecto intenta poner solución a esta problemática ofreciendo un método sencillo mediante el uso de máquinas virtuales desde el navegador.

Motivación

Cuando entré en el grado de ingeniería informática mi idea era acabar haciendo videojuegos, pero no tenia ni la más remota idea de que es lo que aprendería exactamente en el grado. Al ir avanzando cursos pude aprender muchas cosas y viendo el contenido de los cursos y las nefastas condiciones laborales dentro de la industria del videojuego he terminado optando por la opción del desarrollo de software.

Así que Raúl me propuso este proyecto cuando contacté con él me pareció perfecto pues iba en la línea de a lo que me quiero dedicar. Al principio me dio un poco de miedo porque pensaba que quizás no estaba preparado para un reto así. Máquinas virtuales en un navegador, en mi cabeza era algo muy grande. Pero al investigar y ver las tecnologías que veremos en este documento, despertó mi curiosidad el como podría hacer que estas tecnologías interaccionaran entre si para la realización de este trabajo.

Objetivo principal

El objetivo principal de este trabajo el de poder visualizar y utilizar máquinas virtuales en una página web desde el navegador del usuario con la finalidad de poder ofrecer entornos preparados en los que ofrecer ejercicios, como por ejemplo, de ciberseguridad.

Objetivos específicos

El objetivo principal es desglosa en los siguientes objetivos específicos:

- **Investigación de las tecnologías existentes:** Antes de poder plantearme como va a ser el desarrollo del proyecto primero se deberá investigar que tecnologías hay disponibles para poder lograr el objetivo planteado.

- **Diseño de la aplicación:** Una vez estudiadas las posibilidades para la realización del proyecto toca diseñar como va a ser la aplicación.

- **Implementación de la aplicación:** Una vez diseñada la aplicación toca el proceso de programar nuestra página web.

Planificación

Podemos dividir la planificación de este proyecto en tres bloques, representando cada bloque un objetivo específico en el que se desglosaba el principal.

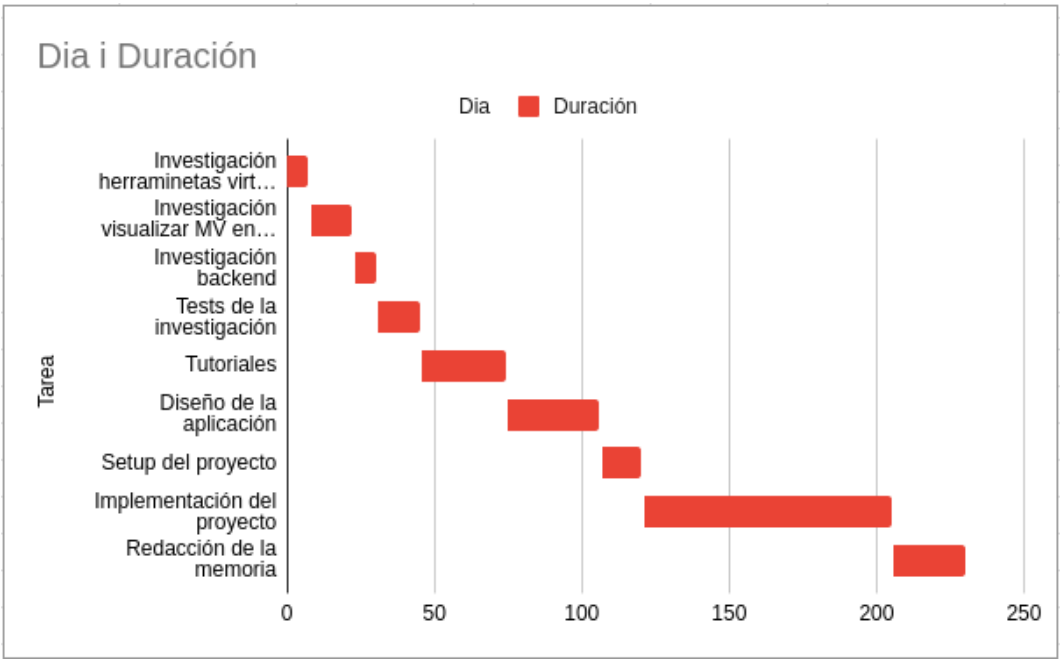
Para poder ver si el proyecto es viable, se dedicarán de Noviembre hasta Enero. Durante estos meses se investigará sobre las tecnologías necesarias para la realización del proyecto, incluso se harán pruebas piloto para poder ir viendo como interactúan entre sí las distintas tecnologías y como pueden ser integradas en un solo programa.

La parte de diseño de la página no debería llevar mucho tiempo, pues en las conversaciones mantenidas con Raúl quedó claro que es lo que se pretende con este trabajo así que para esta parte se dedicará el mes de Febrero.

Finalmente para la implementación de la página tendremos lo que resta de tiempo para la realización del TFG, es decir, de Marzo a Junio. Durante este tiempo se implementará el backend del proyecto, así como se integrarán las distintas tecnologías escogidas para la realización del trabajo.

En la siguiente página se muestra la planificación inicial:

Tarea	Fecha de inicio	Fecha de finalización
Investigación herraminetas virtualización	1.11.2020	8.11.2020
Investigación visualizar MV en navegadores	9.11.2020	23.11.2020
Investigación backend	24.11.2020	1.12.2020
Tests de la investigación	2.12.2020	16.12.2020
Tutoriales	17.12.2020	14.1.2021
Diseño de la aplicación	15.1.2021	15.2.2021
Setup del proyecto	16.2.2021	1.3.2021
Implementación del proyecto	2.3.2021	25.5.2021
Redacción de la memoria	26.5.2021	19.6.2021



Organización del documento

La estructura de esta memoria se desglosa en los siguientes capítulos:

- **Introducción:** En el capítulo de introducción se ha expuesto el contexto de este trabajo y la motivación para hacerlo. También se ha explicado cual es el objetivo principal y la planificación en el tiempo para lograrlo.
- **Análisis:** El capítulo de Análisis expondrá la investigación llevada a cabo y las distintas tecnologías que se han probado para poder llevar a cabo este proyecto.
- **Diseño e Implementación:** En este capítulo se explica los patrones de diseño seguidos así como todo el proceso de diseño. Además se entra más en detalle en el proceso de implementación
- **Simulación de un caso** Este capítulo pretende demostrar de forma práctica como es el flujo de trabajo básico de la aplicación y hacer una pequeña demostración.
- **Conclusiones y trabajo futuro:** En este capítulo se detalla las conclusiones extraídas de este proyecto y el trabajo que queda a realizar.
- **Apéndice: Manual Técnico:** En el Manual Técnico se especifican las dependencias del proyecto, así como las versiones utilizadas de cada tecnología y la configuración necesaria para poder ejecutarlo.

Capítulo 1

Análisis

1.1. Introducción

En este capítulo se expone todo lo relacionado con la investigación realizada sobre las tecnologías necesarias para llevar a cabo el proyecto y si es posible llevarlo a cabo. Este capítulo lo dividiremos en tres bloques que son los siguientes.

- **Elección de sistema operativo:** Tendremos en cuenta los motivos por los cuales me decido por un sistema operativo en concreto y no por otro.
- **Investigación sobre tecnologías de virutalización:** En este punto compararemos las tecnologías existentes para la virtualización de máquinas virtuales y explicaré por que me decanto por una opción u otra.
- **Investigación sobre visualización de maquinas virtuales en navegadores:** Aquí veremos cual fue el proceso de investigación para la selección de la tecnología que nos permita llevar a cabo el objetivo principal del trabajo.
- **Investigación de tecnologías para el backend (frameworks, lenguajes, etc):** Una vez realizada toda la investigación y seleccionadas las herramientas tendremos que escoger cual es el lenguaje de programación que mejor se adapta a todas ellas y nos permita comunicarlas entre si.
- **Conclusiones:** Al terminar los puntos anteriores tendremos que valorar si realizar el proyecto es posible.

1.2. Elección de sistema operativo

La elección de sistema operativo fue relativamente fácil. Sin querer innovar demasiado me decidí por Ubuntu 20. Debido a experiencias anteriores con distintos sistemas operativos he podido comprobar que a la hora de virtualizar maquinas virtuales, en Ubuntu corren ligeramente mejor que en por ejemplo Windows 10, sobretodo cuando se tiene más de una abierta a la vez, y puesto que no dispongo de ningún dispositivo con MacOS con el cual comparar me quedo con Ubuntu.

1.3. Investigación sobre tecnologías de virutalización

Cuando tenemos que mirar por opciones para virtualizar máquinas tenemos dos grandes opciones en distribuciones de Linux: VMWare o Virtualbox. Teniendo en cuenta la siguiente tabla[1]:

	Oracle	VMware				
	VirtualBox 5.2.22	Player 15	Workstation 15	Fusion 11	Fusion Pro 11	ESXi 6.7
Hypervisor Type	2	2	2	2	2	1
Licensing	Free	Free	From 250 $	80 $	160 $	Free/Paid (from 495$)
Virtualization	Hardware + Software	Hardware	Hardware	Hardware	Hardware	Hardware
Host OS	Linux, Windows, Solaris, macOS, FreeBSD	Linux, Windows	Linux, Windows	macOS	macOS	---
Guest OS	Linux, Windows, Solaris, FreeBSD, macOS	Linux, Windows, Solaris, FreeBSD	Linux, Windows, Solaris, FreeBSD	Linux, Windows, Solaris, FreeBSD, macOS	Linux, Windows, Solaris, FreeBSD, macOS	Linux, Windows, Solaris, FreeBSD
Shared Folders	Yes	Yes	Yes	Yes	Yes	No
Seamless mode/Unity	Yes	Yes	Yes	Yes	Yes	No
VM snapshots	Yes	No	Yes	Yes	Yes	Yes
USB for VMs	With Extension Pack	Out of the box	Out of the box	Out of the box	Out of the box	Out of the box
3D graphics in VMs	DirectX 9, OpenGL 3.0	DirectX 10, OpenGL 3.3	DirectX 10, OpenGL 3.3	DirectX 10, OpenGL 3.3	DirectX 10, OpenGL 3.3	DirectX 10, OpenGL 3.3
Max.VM video memory	128 MB	2 GB	2 GB	2 GB	2 GB	2 GB
Virtual Disk Format	VDI, VMDK, VHD, HDD*	VMDK	VMDK	VMDK	VMDK	VMDK
Linked clones support	Yes	No	Yes	No	Yes	No*
Shared storage support	iSCSI, NFS, SMB (CIFS)	No*	No*	No*	No*	iSCSI, NFS, Fibre Channel
VM Live Migration	Yes (Teleportation)	No	No	No	No	Yes (vMotion)
Centralized Management	PhpVirtualBox	No	No	No	No	vCenter
VM Encryption	Yes, with Ext. Pack	Yes (limited)	Yes	Yes (limited)	Yes	Yes
Memory Ballooning	Yes	-	Yes	Yes	Yes	Yes
Clustering	No	No	No	No	No	Yes

Vamos a fijarnos en los siguientes apartados y los trataremos de forma individualizada: licencia, sistema operativo anfitrión e invitado, memoria de vídeo máxima y networking.

1.3.1. Licencia

En el caso de Virtualbox existe una sola licencia[8] que es gratuita en el caso de usar el Virtualbox base. Sin embargo existe el Virtualbox Extension Pack, que incluye funcionalidades como el soporte de USB 2.0 y 3.0, RDP o la encriptación del disco de la máquina virtual que son gratuitas si son utilizadas para fines personales o educacionales, pero en caso de querer sacar rédito económico con estas funcionalidades habría que pagar una licencia que puede ir de los 540 euros a los 1080 euros según el plan deseado.

Por otra parte, VMWare[9] tiene más opciones que Virtualbox pero la versión gratuita Player 15 viene limitada en comparación con su rival y versiones con más funcionalidades, ya sea para uso personal, educacional o empresarial requiere el pago de una licencia que varia desde los 200 euros para las versiones más básicas a los 4900 euros aproximadamente en caso de las versiones que incluyen soporte.

1.3.2. Sistema operativo anfitrión e invitado

Como vemos en la gráfica ambas herramientas de virtualización pueden ser usadas en sistemas operativos Windows y Linux, y tambíen pueden virtualizar sistemas operativos de Windows o distribuciones de Linux.

1.3.3. Memoria de vídeo máxima

En este apartado podemos observar como VMWare nos ofrece una mayor cantidad de memória de video máxima, llegando así a los 2 GB. En el caso de Virtualbox esta memoria de vídeo apenas llega a los 128 MB. También cabe destacar que en cuanto a las tecnologías en gráficos 3D, VMWare utiliza versiones más nuevas que las de Virtualbox.

1.3.4. Networking

Aunque en la gráfica no aparezca esta sección me parece necesario mencionar las posibilidades que nos ofrece cada una de las opciones. Virtualbox nos ofrece:

- **Not attached:** Con esta opción Virtualbox emulará un cable Ethernet desconectado, por lo tanto la máquina virtual no tendrá conexión a internet.
- **NAT:** Este modo permite habilitar una red privada para una máquina virtual conectada a un router virtual integrado en el motor de Virtualbox. Una máquina invitada puede conectarse a la máquina anfitrión y a otras máquinas virtuales en la red. Las máquinas virtuales conectadas a la NAT tendrán acceso a internet y un servidor DHCP integrado asignará las direcciones IP a los adaptadores de red de cada máquina virtual.
- **NAT Network:** Es un modo más avanzado que el NAT y requiere de la creación manual de una red desde las configuraciones de Virtualbox

- **Bridged Adapter:** Este modo es como si varios dispositivos estuviesen conectados por cables al mismo switch.
- **Internal Network:** Este modo permite a las máquinas virtuales conectarse entre sí. Sería como si las máquinas virtuales estuviesen conectadas a un mismo switch, pero éste no estuviese conectado a ningún router, por lo tanto en este modo, las máquinas virtuales no tendrían conexión a internet.
- **Host-only Adapter:** En este modo las máquinas virtuales se conectan a una red aislada donde se pueden comunicar entre ellas incluso con el host. Se crea un adaptador de red en el sistema operativo anfitrión.

Por la otra parte, VMWare nos ofrece:

- **NAT**
- **Bridged**
- **Host-only**

Estos modos de VMWare funcionan de forma muy similar a sus homólogos de Virtualbox.

1.3.5. Elección de herramienta de virtualización

Habiendo hecho un repaso a los características que nos ofrece cada herramienta, al final me decidí por Virtualbox. La versión gratuita me daba más opciones que su contraparte gratuita de VMWare. Aunque la memoria de vídeo está mucho más limitada en Virtualbox que en VMWare creo que no es será tan importante para este proyecto pues en principio no se busca hacer nada que sea computacionalmente caro en cuanto al uso de la memoria de vídeo, simplemente visualizar un sistema operativo.

1.4. Investigación sobre visualización de maquinas virtuales en navegadores

Habiendo elegido ya Virtualbox sobre VMWare me dispuse a investigar sobre las tecnologías ya existentes para poder visualizar una máquina virtual en el navegador.

1.4.1. PhpVirtualbox

Investigando sobre como gestionar maquinas virtuales desde un navegador me encontré con esta solución específica para virtualbox: PhpVirtualbox[4]. Es un frontend para Virtualbox que deja manejar la interfaz gráfica desde el navegador. Pensé que de alguna manera podría hacer que solo mostrara la máquina virtual pero este programa este hecho específicamente para controlar una instalación de Virtualbox que se encuentre en otro ordenador sin salida de vídeo, de difícil acceso o bien de forma remota desde casa, pero en ningún caso poder abrir instancias de las máquinas virtuales para operarlas de forma separada (y por diferentes usuarios de forma simultánea).

1.4.2. Acceder de forma remota a un ordenador

Después de descartar la opción de phpVirtualbox, seguí investigando y encontré protocolos que sirven para conectarse y utilizar ordenadores de forma remota. Los dos protocolos más usados son RDP (Remote Desktop Protocol) y VNC (Visual Networking Computing). Ambos protocolos solucionarían el problema de poder conectarme a una maquina virtual y operarla remotamente pero vamos a ver las similitudes y diferencias para compararlos[6].

Similitudes:

- Ambos comparten la misma meta. Acceder remotamente a otro ordenador, mostrando el escritorio, comunicar las acciones del teclado y ratón.
- Ambas tecnologías usan comunicación peer-to-peer. Se puede capar la conexión mediante firewall. En este caso la comuniación se puede establecer mediante un ordenador intermediario.
- Ambas tecnologías requieren de un cliente y un servidor para establecer la comunicación.

Diferencias:

- RDP se conecta en un usuario remoto al servidor creando una sesión de escritorio real. Es como si un usuario se hubiese conectado en el servidor físicamente. RDP puede tener varios usuarios remotos conectados al mismo servidor sin que sepan nada de los otros usuarios. Es especialmente útil si se quiere tener a varios usuarios conectados al mismo servidor de forma remota.

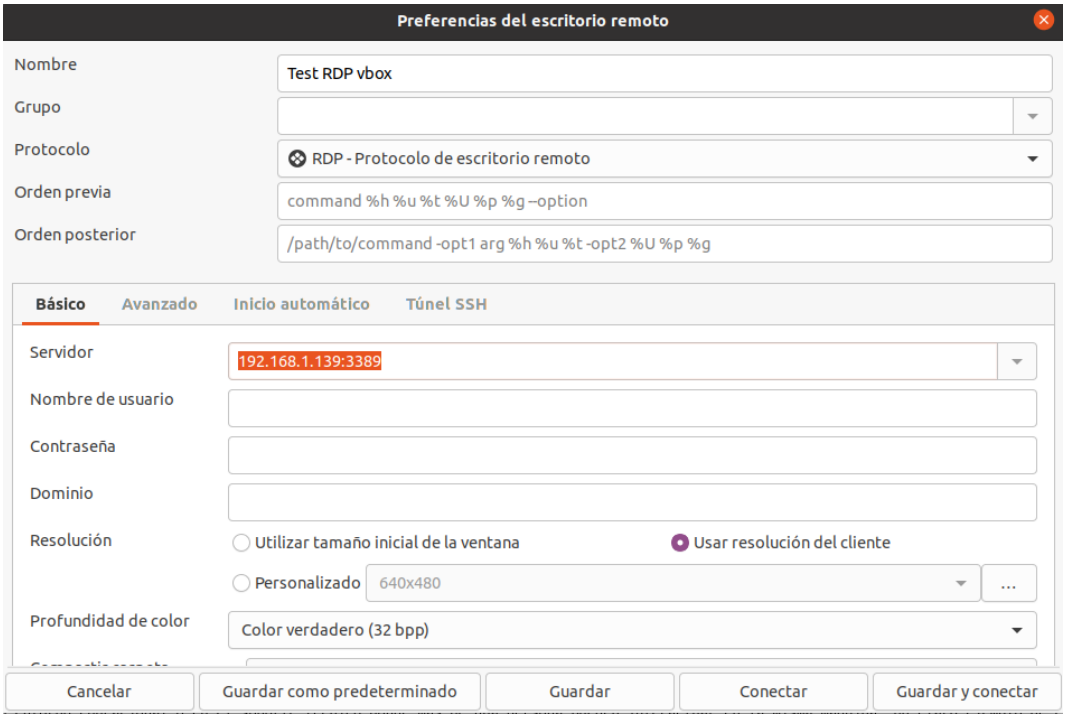
- VNC se conecta a un usuario remoto compartiendo su pantalla, teclado y ratón. Consecuentemente, cuando varios usuarios, incluyendo el de la máquina física, se conectan a la misma máquina física ven lo mismo y comparten el mismo teclado.
- VNC es un protocolo pixel-based, es decir, envía imágenes a través de la red en forma de imágen haciendo que pierda rendimiento. En cambio RDP transforma la información gráfica generada en el servidor en datos que entiende el protocolo y transforma la imagen en el lado del cliente, evitando así enviar imágenes a través de la red y ahorrando recursos.

Aparte de estas diferencias, hay que tener en cuenta que RDP es un protocolo propietario de Windows y en caso de trabajar en ese sistema operativo es bastante recomendable, aunque hay clientes disponibles en distribuciones Linux que permiten usar este protocolo sin ningún tipo de problema.

VNC está más orientado a un entorno educacional o en el soporte técnico donde más de una persona pueden interactuar en la misma máquina. Un claro ejemplo de esto es el programa Team Viewer.

Visto esto creo que será más útil conectarnos a una máquina virtual vía RDP, pues usando VNC estaríamos desperdiciando uno de sus principales casos de uso. Ahora veamos si podemos conectarnos a una máquina virtual usando este protocolo. Para ello utilizaré un cliente de pantalla remota que viene ya instalado en Ubuntu llamado Remmina[7].

El primer paso es crear una conexión y en se nos abrirá un formulario donde introducir los datos. Pondremos RDP como protocolo y en servidor la IP del ordenador seguido de dos puntos y el puerto en el que se ejecuta el servidor de pantalla remota, en este caso el puerto por defecto es 3389. La configuración en Remmina quedaría así:



1.4.3. Virtualbox SDK

Viendo que al final utilizaré Virtualbox y de que necesitaré poder manejar las máquinas virtuales de forma programática, por ejemplo, encenderlas, apagarlas, conectarlas entre sí, tuve que buscar alguna manera de poder hacerlo. La solución a este problema ha sido el SDK (Software Development Kit) de Virtualbox. Para operar las máquinas el SDK permite conectarse a un webservice que se puede correr en el ordenador, y una vez conectado el SDK con el webservice se podrá interactuar con las máquinas virtuales.

La conexión al webservice que viene con la instalación de Virtualbox viene definida en varios lenguajes de programación populares, siendo estos Java, Python, PHP y C++.

1.4.4. Apache Guacamole

Teniendo en cuenta la sección anterior y viendo que podía conectarme a una maquina virtual usando el protocolo RDP, encontré un cliente de pantalla remota para HTML5, Apache Guacamole, por lo que podía ser integrado en un servidor web, open-source y gratuito.

Para instalar Apache Guacamole en un sistema se necesita descargar el servidor y cliente que proporcionan en su página web. También hay que instalar un programa daemon, es decir que se ejecuta en el background, y que se encarga de aceptar las peticiones de los usuarios y de manejar los túneles que conectan el servidor con las máquinas virtuales. Al principio, para probar como iba instalé el servidor en un servidor

Tomcat 9 e instalé el cliente. Una vez se tiene montado el sistema, es muy similar a lo que se ha explicado con Remmina, se introducen los datos necesarios como el protocolo a utilizar, puerto e IP y el sistema te deja manejar desde el navegador la maquina virtual. Para hacerle la vida más fácil a los desarrolladores han puesto a disposición de todos las APIs que utilizan para hacer esto posible, que son las siguientes:

- **libguac:** Libreria escrita en C que sirve para desarrollar y extender Apache Guacamole y se usa principalmente para desarrollar clientes como el cliente para RDP o VNC
- **guacamole-common:** API escrita en Java. Provee de métodos básicos para pasar la información del lado del servidor al cliente JavaScript de Guacamole para la visualización de las máquinas virtuales en el navegador.
- **guacamole-common-js:** Sirve de interfaz para proyectos que usan libguac y/o guacamole-common. Provee de mecanismos de abstracción para capturar teclado y ratón y métodos para preparar los datos del protocolo y ser enviados al servidor y guacd.
- **guacamole-ext:** Aunque no es parte de la API de Java, sirve para poder integrar bien guacamole en proyectos ya existentes.

Solo con guacamole-common en el lado del servidor y guacamole-common-js en el frontend, deberíamos de ser capaces de poder conectarnos a una máquina virtual desde el navegador.

1.5. Investigación de tecnologías para el backend

1.5.1. Lenguaje de programación

Como la API guacamole-common de Apache Guacamole está escrita en Java y el SDK de Virtualbox se puede elegir la implementación en el mismo lenguaje, creo que la elección más inteligente seria escoger Java para desarrollar este trabajo de fin de grado.s

1.5.2. Framework para el backend

Para escoger framework de backend para el proyecto y no tener que pelearme a demasiado bajo nivel, mi elección ha sido usar Spring Framework, con muchos recursos en la red y una comunidad grande.

1.6. Conclusiones

Después de realizar pruebas para ver como interaccionaban todas las herramientas entre sí, la configuración del proyecto ha quedado de la siguiente forma:

- **Sistema Operativo:** Por comodidad a la hora de desarrollar, Ubuntu 20.04.
- **Herramienta de Virtualización:** Virtualbox porque la licencia gratuita supera con creces a la de su competidor.
- **Visualización de máquinas virtuales en navegadores:** Apache Guacamole es el framework que utilizaré
- **Tecnología para el backend:** Por dependencias con el SDK de Virtualbox y la librería guacamole-common usaré Java y de framework Spring.

Capítulo 2

Diseño e Implementación

En este capítulo habrá dos apartados principales. En el primero veremos el diseño de la aplicación: decisiones tomadas, diagrama relacional de la base de datos, diagramas de clase para ver como interactúan las clases entre ellas. En el segundo apartado veremos como han sido implementadas distintas funciones de la aplicación de forma más detallada.

2.1. Diseño de la aplicación

2.1.1. Patrones de diseño

Modelo-Vista-Controlador

Spring Framework trabaja usando el patrón de diseño de Modelo-Vista-Controlador. Este patrón de diseño separa la lógica del programa en tres elementos interconectados. De esta manera se logra separar la forma en la que se presentan los datos de como se manipulan de como se gestionan los eventos. Estas tres partes son:

- **Modelo:** Contiene la representación de la información y se encarga de hacer operaciones sobre los datos, de la lógica de la aplicación y de los permisos implementados.

- **Controlador:** Responde a eventos, como pueden ser acciones del usuario y los convierte en peticiones al modelo para pedir información o que haga una transformación sobre estos.

- **Vista:** Presenta el modelo al usuario, en nuestro caso la vista son los html que presentamos al usuario.

Singleton

Este patrón de diseño se encarga de restringir la instanciación de un objeto a un por aplicación para poder trabajar con la misma instancia en todo el programa. En el caso de Spring, este patrón no se aplica a la aplicación entera, si no que se restringe la instanciación de un objeto por cada contenedor IoC (Inversion of Control) también conocido como Inyección de Dependencias. En Spring usaremos la anotación @Component para registrar una clase como un Bean (almacenará donde se definirán las dependencias, es decir, con que otros objetos trabaja) y con la anotación @Autowired Spring inyectará el Bean en un atributo de un objeto.

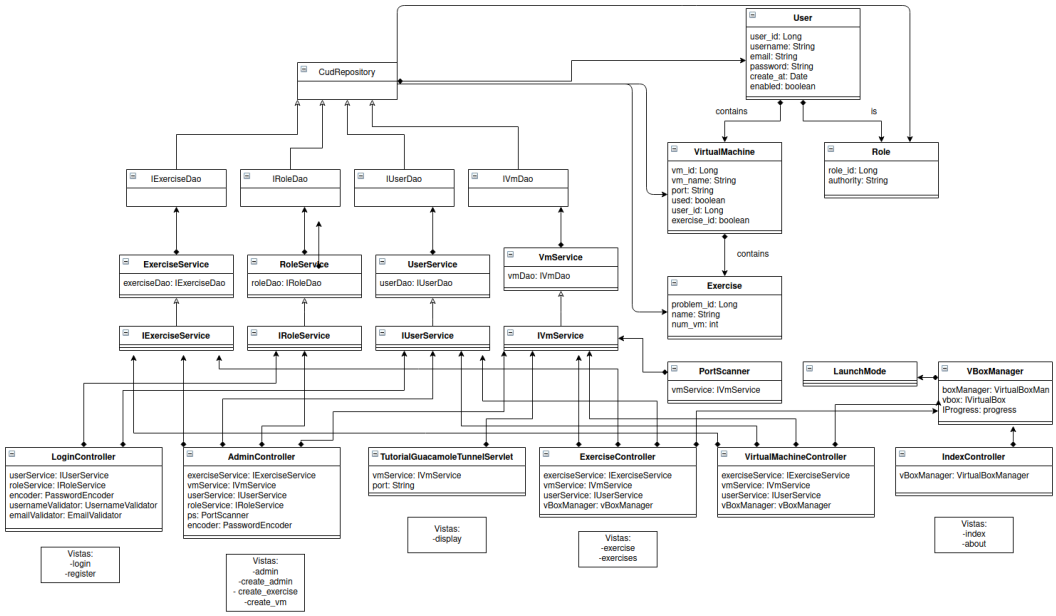
Factory method pattern

Es un patrón que soluciona el problema de crear objetos sin tener que especificar la clase exacta del objeto. Esto se consigue mediante llamadas a factory methods en vez de llamar a los constructores. Es especialmente útil cuando se necesita construir un objeto que tiene un atributo que es una interfície pero no se sabe aún de que tipo exacto va a ser.

Spring utiliza este patrón en la raíz de su Inyección de Dependencias para producir los Beans y tratará el contenedor de Beans como una factory.

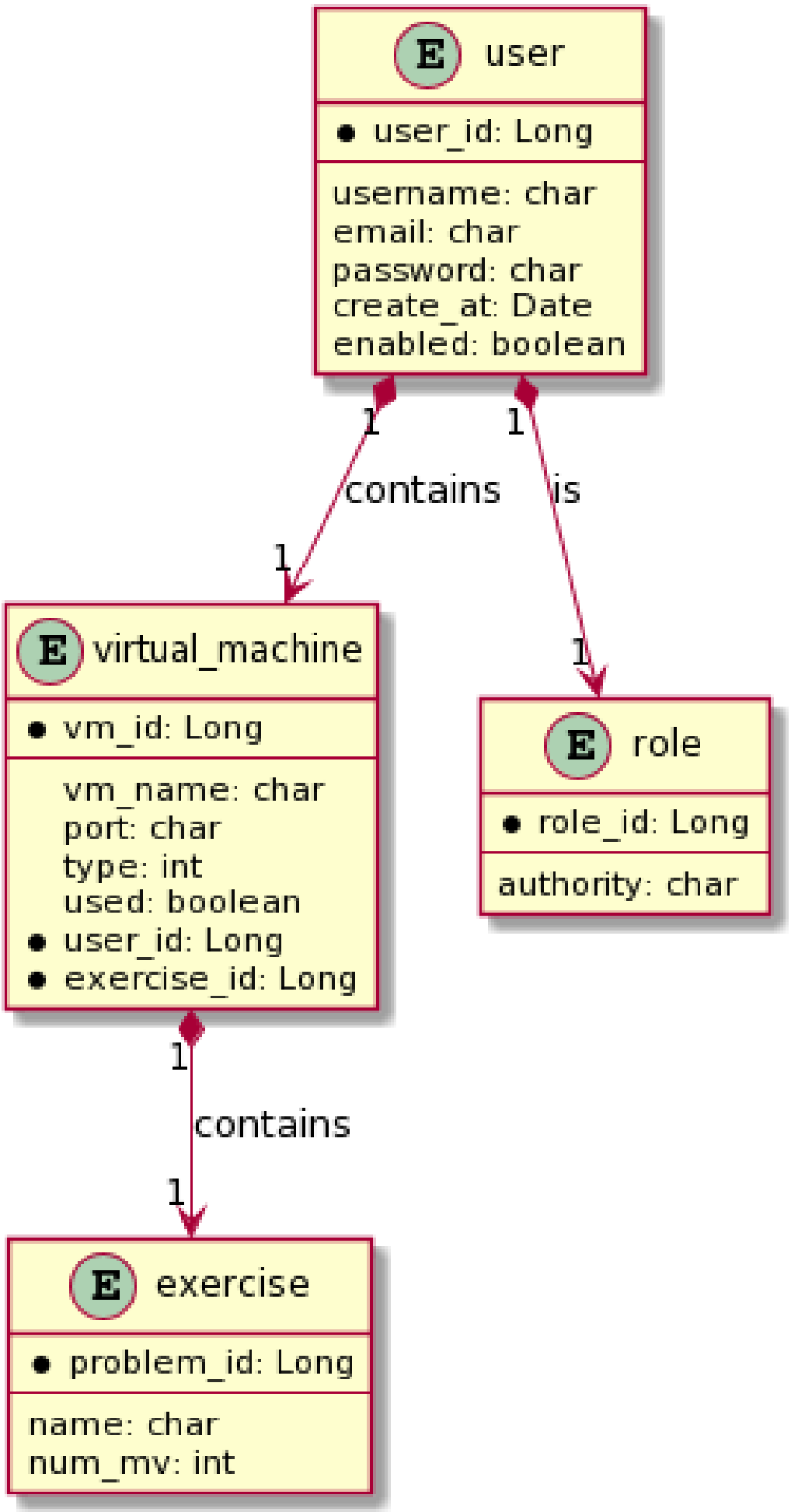
2.1.2. Diagrama de clases

A continuación se adjunta el diagrama de clases:



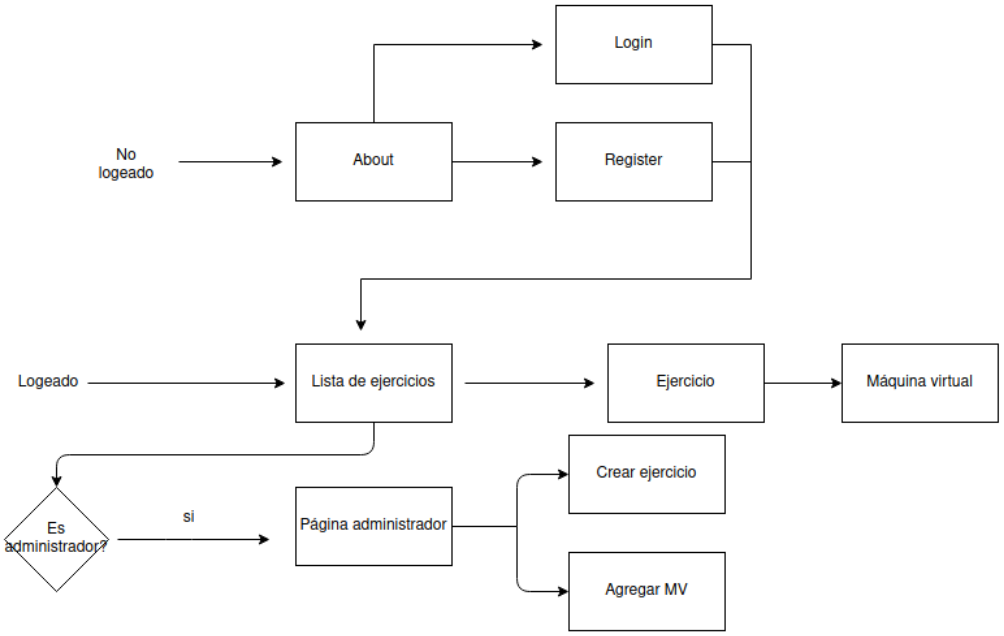
2.1.3. Diagrama de la base de datos

En este apartado se adjuntará un diagrama con las tablas de la base de datos y como están relacionadas:


user
user_id: Long
username: char
email: char
password: char
create_at: Date
enabled: boolean
1
contains
1
1
is
1
virtual_machine
vm_id: Long
vm_name: char
port: char
type: int
used: boolean
user_id: Long
exercise_id: Long
role
role_id: Long
authority: char
1
contains
1
exercise
problem_id: Long
name: char
num_mv: int

2.1.4. Diagrama de flujo

En este apartado se adjuntará un diagrama de flujo para entender el funcionamiento básico de la aplicación con una sola mirada.



2.2. Implementación

En esta sección veremos la solución e implementación de algunos problemas interesantes que me he ido encontrando a lo largo del desarrollo del proyecto, aunque no entraremos en cosas básicas del framework Spring. Dividiremos esta sección en varios apartados siendo estos: Máquinas virtuales, Guacamole y

2.2.1. Configuración CSRF Spring

Al añadir la protección contra CSRF (Cross-Site Request Forgery) la visualización de las máquinas virtuales dejó de funcionar. Después de investigar descubrí cual era el motivo. En la configuración de Spring, tuve que añadir este trozo de código para que desactivara la protección para la url /tunnel.*", es decir, para todos los patrones que empezaran por /tunnelz se ignora lo que viene después, que en el caso de guacamole son datos del protocolo que usa para comunicar el navegador con el frontend.

```
.requireCsrfProtectionMatcher(new RequestMatcher() {
    private final Pattern allowedMethods = Pattern.compile(
                                                "^(GET|HEAD|TRACE|OPTIONS)$");
```

```
        private final RegexRequestMatcher guacamoleMatcher = new RegexRequestMatcher(
                                                    "/tunnel.*", null);

        @Override
        public boolean matches(HttpServletRequest request) {
            // Disable csrf in allowed methods
            if(allowedMethods.matcher(request.getMethod()).matches()) {
                return false;
            }
            // No csrf due to connection with guacamole
            if(guacamoleMatcher.matches(request)){
                return false;
            }
            return true;
        }
    })
```

2.2.2. Autenticación

Para autenticar los usuarios mediante un inicio de sesión, se necesita una contraseña. El método de autenticación que decidí para la aplicación fue mediante base de datos. Para ello las credenciales se guardan en una tabla de la base de datos. Pero como no se debe de guardar la contraseña como texto plano, las contraseñas se encriptan usando el algoritmo de BCrypt antes de ser almacenadas en la base de datos.

2.2.3. Máquinas virtuales

Crear máquina virtual

La primera solución que intenté implementar fue la de clonar máquinas virtuales ya existentes. El principal problema es que el proceso de clonación es largo y hacía que el servidor quedara inoperativo. Por este motivo opté por otra opción: tener una pool de máquinas virtuales ya creadas en el servidor y que un usuario administrador pudiera elegir una máquina mediante un formulario usando el nombre para encontrarla y asignarle un ejercicio y función que desempeña en el ejercicio.

Encontrar un puerto libre

Para poder visualizar la máquina virtual en nuestro navegador necesitamos encontrar primero un puerto que esté libre. Cuando se añade una máquina virtual al sistema, hay un proceso el cual busca un puerto que no esté siendo usado ya por ninguna otra máquina virtual o bien por ningún otro proceso en la máquina en la que corre.

```
public String getAvailablePort(){
    String port = "";
```

```
    for(int i = 1024; i < 65535; i++){
        String port_str = Integer.toString(i);
        if(!isPortInDB(port_str) && !isPortInUse("localhost", i)){
            port = port_str;
            break;
        }
    }
    return port;
}
```

La función isPortInDB busca que en la base de datos ese puerto no esté siendo usado por ninguna máquina virtual comparándolo con los puertos ya almacenados en la base de datos.

```
private boolean isPortInDB(String port){

    List<VirtualMachine> vms = vmService.findAll();

    if(!vms.isEmpty()){
        for(VirtualMachine vm : vms){
            if(vm.getPort().equals(port)){
                return true;
            }
        }
    }
    return false;
}
```

Y el método isPortInUse mirará que no esté siendo usada por el sistema. Para ello abriremos un Socket en el puerto deseado y lo cerraremos al instante, dentro de bloque try-catch. Si no falla signficará que el puerto está libre y hará que el retorno sea true.

```
private boolean isPortInUse(String host, int port) {

    // Assume no connection is possible.
    boolean result = false;

    try {
        (new Socket(host, port)).close();
        result = true;
    }
    catch(Exception e) {
        // Could not connect.
    }

    return result;
}
```

DHCP Server

En muchos casos será necesaria la comunicación entre máquinas virtuales. Para ello la solución que ha sido implementada es la de crear un servidor DHCP usando el sdk de virtualbox. Este servidor DHCP solo se creará cuando se entre en un ejercicio donde sean necesarias más de dos máquinas virtuales.

```
// If the necessary Vms to carry out the exercise is greater than one we have to create
if(numVms > 1){
    List<String> vmNames = new ArrayList<>();
    for(VirtualMachine vm: vms){
        vmNames.add(vm.getVm_name());
    }
    DHCPServer dhcpConf = new DHCPServer();
    String dhcp_name = actualExercise.getName().concat(".").concat(principal.getName());
    dhcpConf.autoconfigure(dhcp_name);
    vBoxManager.assignInternalNetworkFromMachineNames(dhcp_name ,vmNames, dhcpConf);
}
```

Primero extraeremos los nombres de las máquinas virtuales porque es el parámetro que requiere el sdk de virtualbox para encontrar las máquinas virtuales. Luego crearemos un objeto DHCPServer que con el método autoconfigure pondremos una configuración básica que es la siguiente. El único valor variable es el nombre que es la unión del nombre del ejercicio junto con el nombre del usuario.

```
public void autoconfigure(String name){
    this.setNetname(name);
    this.setUsed(true);
    this.setIp("10.0.0.1");
    this.setNetMask("255.255.255.0");
    this.setLower_ip("10.0.0.2");
    this.setUpper_ip("10.0.0.12");
}
```

En éste código estamos diciendo que la IP 10.0.0.1 será la IP del servidor DHCP y que a las máquinas virtuales se les asignará una IP que se encuentre entre la 10.0.0.2 y la 10.0.0.12. La máscara de red nos indica el rango de las IPs. En este caso podrían ir desde la 10.0.0.1 hasta la 10.0.0.255 pero ha sido capado hasta la 10.0.0.12 porque difícilmente vamos a encontrar un problema que requiera de tantas máquinas virtuales por no decir imposible.

Ahora vamos a ver el método assignInternalNetworkFromMachineNames.

```
public boolean assignInternalNetworkFromMachineNames(String networkName,
    List<String> machineNames, DHCPServer conf){

    boolean fail = false;

    if(machineNames.size() < 2){
        return fail;
```

```
        }

        if(existsDHCPServer(networkName)){
            deleteDHCPServer(networkName);
        }

        createDHCPServer(networkName, conf);


        for(String machineName: machineNames){
            if(machineExists(machineName)) {
                IMachine machine = findMachine(machineName);
                ISession session = boxManager.getSessionObject();
                machine.lockMachine(session, LockType.Write);
                IMachine mutable = session.getMachine();
                INetworkAdapter networkAdapter = mutable.getNetworkAdapter(0L);
                networkAdapter.setAttachmentType(NetworkAttachmentType.Internal);
                networkAdapter.setInternalNetwork(networkName);
                networkAdapter.setPromiscModePolicy(NetworkAdapterPromiscModePolicy.AllowNetw
                mutable.saveSettings();
                session.unlockMachine();
            }else {
                fail = true;
                break;
            }
        }

        return fail;
    }
```

En este algoritmo primero se hace una comprobación de que haya como mínimo más de una máquina virtual. Después se comprueba de que exista ya un servidor DHCP con el mismo nombre y si existe se borra. Entonces ya podemos crear el servidor DHCP. Aquí podemos ver como funcionan los métodos mencionados:

```
public boolean existsDHCPServer(String name){

    boolean exists = false;
    for(IDHCPServer dhcp : vbox.getDHCPServers()){
        if(dhcp.getNetworkName().equals(name)){
            exists = true;
            break;
        }
    }
    return exists;
}

public IDHCPServer findDHCPServerByName(String name){
    for(IDHCPServer dhcp : vbox.getDHCPServers()){
        if(dhcp.getNetworkName().equals(name)){
            return dhcp;
        }
    }
    return null;
}

public void createDHCPServer(String name, DHCPServer conf){
    IDHCPServer dhcpServer = vbox.createDHCPServer(name);
    // IpAddress, networkMask, lowerIp, upperIp
    dhcpServer.setConfiguration(conf.getIp(), conf.getNetMask(), conf.getLower_ip(), con
    dhcpServer.setEnabled(true);
    dhcpServer.start(name, String.valueOf(NetworkAttachmentType.Internal));
}

public void deleteDHCPServer(String name){
    IDHCPServer toRemove = findDHCPServerByName(name);
    if(toRemove != null){
        vbox.removeDHCPServer(toRemove);
    }
}
```

Sobre el código del for hablaremos en un apartado posterior donde se mostrará cómo modificar las características de una máquina virtual.

Modificar máquina virtual

Cuando se crea una máquina virtual hay que modificarla para poner los datos que nosotros queramos. Este es el caso que nos encontramos al poner un puerto para el servidor de pantalla remota o cuando hay que conectar las máquinas virtuales mediante el servidor DHCP. Pero hay que tener en cuenta que de la misma forma que no podemos cambiarle la memoria RAM o la capacidad de almacenamiento a un ordenador mien-

tras está encendido, cuando se vaya a modificar algún valor de la configuración de la máquina virtual ésta deberá estar apagada.

Poner puerto de servidor de pantalla remota

```
public void setPortVRDE(String nameMachine, String port){
    IMachine machine = findMachine(nameMachine);
    ISession session = boxManager.getSessionObject();
    machine.lockMachine(session, LockType.Write);
    IMachine mutable = session.getMachine();
    mutable.getVRDEServer().setVRDEProperty("TCP/Ports", port);
    mutable.getVRDEServer().setEnabled(true);
    mutable.saveSettings();
    session.unlockMachine();

}
```

Este es un perfecto ejemplo de como se modifica un valor en la configuración de una máquina virtual. De forma normal, aunque si se puede hacer un get de cualquier valor de la máquina virtual, la escritura está bloqueada. Esto es debido a que el entorno de virtualbox es un entorno complicado donde muchos procesos pelean por posiblemente los mismos recursos. Por este motivo, las máquinas virtuales deben de estar en el estado 'locked' antes de ser modificadas o encendidas. Así se evita que dos o más procesos hagan cambios a las configuraciones a la vez o que enciendan varias máquinas virtuales a la vez.

El primer paso es encontrar la máquina virtual que se quiere modificar. Después crearemos un objeto ISession. Cada llamada que hagamos al IWebSessionManager (en el código es el objeto boxManager), creará un objeto de ISession nuevo. Este objeto se usará como un si de un semáforo mutex se tratara, una vez se intente entrar en un bloque de código después del método lockMachine bloqueará cualquier otro intento de cualquier proceso que intente iniciar el proceso de modificación de una máquina virtual. Este objeto ISession cuando es pasado al método lockMachine, hará una copia de todos los atributos de la máquina virtual original y todos los cambios que se hagan con los métodos set se guardarán en un objeto IMachine, copia de la original. Cuando se ejecute el método saveSettings, se guardarán todos los cambios realizados en el objeto IMachine de la ISession directamente a la máquina virtual original. Una vez realizados todos los cambios simplemente hay que llamar al método unlockMachine para desbloquear la máquina.

Para obtener la variable de la configuración a modificar primero hay que obtener el objeto que representa las propiedades a modificar. Después, en vez de haber un set por propiedad, en el método set hay que pasar una string que represente el valor que se quiere modificar. En este caso concreto queremos cambiar el puerto del Remote Desktop Extension"

Conectar máquinas virtuales a servidor DHCP

```
for(String machineName: machineNames){
    if(machineExists(machineName)) {
        IMachine machine = findMachine(machineName);
        ISession session = boxManager.getSessionObject();
        machine.lockMachine(session, LockType.Write);
        IMachine mutable = session.getMachine();
        INetworkAdapter networkAdapter = mutable.getNetworkAdapter(0L);
        networkAdapter.setAttachmentType(NetworkAttachmentType.Internal);
        networkAdapter.setInternalNetwork(networkName);
        networkAdapter.setPromiscModePolicy(NetworkAdapterPromiscModePolicy.AllowNetwork
        mutable.saveSettings();
        session.unlockMachine();
    }
```

De igual forma que se ha explicado en el apartado anterior, en este trozo de código se itera por los nombres de las máquinas virtuales, se crea un objeto ISession y se bloquea la máquina virtual. Se obtiene el objeto IMachine modificable y obtenemos el adaptador de red. A diferéncia del anterior caso, hay un set por cada variable. Se ha decidido que el tipo de adaptador será de tipo Internal, es decir, las máquinas podrán comunicarse entre si pero no podrán conectarse a Internet. Se le asigna el servidor DHCP usando el nombre dado al servidor. Al terminar se desbloqueará la máquina.

2.2.4. Guacamole

En este apartado se explicará como se ha integrado guacamole con Spring Framework para poder visualizar las máquinas virtuales en el navegador

Preparando los datos

Para poder visualizar más de una máquina virtual en el navegador se necesita poder poner el puerto de forma dinámica. Para ello, cuando se clica el link de una máquina virtual, llamará a esta función que la encenderá. En la request se añadirá el atribute del puerto y se hará un forward a la página encargada de visualizar las máquinas virtuales. Se ha decidido hacer un forward en vez de un redirect por dos motivos: el primero es que el forward es invisible al navegador y se hace en el lado de servidor, por lo tanto la url no cambia. El segundo es porque los atributos de la request se comparten y puedo pasar de forma sencilla el puerto y el nombre de la máquina virtual.

```
@GetMapping("/mv")
public String openVM(@RequestParam String machineName, Model model, HttpServletRequest r
    vBoxManager.launchMachine(machineName, LaunchMode.headless);
    model.addAttribute("titulo", machineName);
    String port = vmService.findByMachineName(machineName).getPort();
    request.setAttribute("port", port);
```

```
        request.setAttribute("machineName", machineName);
        return "forward:/display";
```

Integración con Spring

En el método displayVm que se ejecuta cuando se hace una request a la url '/display' se recupera el puerto que se envia en el método explicado anteriormente y se asigna al atributo de la clase. Los otros dos métodos: handleTunnelRequest y doConnect son métodos heredados de la clase GuacamoleHTTPTunnelServlet que viene dentro de la extensión de guacamole guacamole-common. Cuando desde el frontend se hace una petición para conectarse a guacamole, se ejecuta la función doConnect, que se sobreescribe para que se pueda poner la configuración de túnel deseada entre guacd y el navegador. El método handleTunnelRequest de forma normal no se tiene que heredar pero en este caso ha sido necesario para integrarlo con Spring Framework, para que se ejecute la función con cada petición GET y POST a la url '/tunnel', usada por el protocolo de guacamole para enviar intercambiar información con el cliente y guacd.

```
@Controller("/tunnel")
public class TutorialGuacamoleTunnelServlet extends GuacamoleHTTPTunnelServlet {

    @Autowired
    private IVmService vmService;
    private String port;

    @Override
    @RequestMapping(path = "tunnel", method = { RequestMethod.POST, RequestMethod.GET })
    protected void handleTunnelRequest(HttpServletRequest request, HttpServletResponse respon
        super.handleTunnelRequest(request, response);
    }

    @GetMapping("/display")
    public String displayVm(Model model, HttpServletRequest request){
        String port = (String) request.getAttribute("port");
        this.port = port;
        model.addAttribute("titulo", (String) request.getAttribute("machineName"));
        model.addAttribute("port", port);

        return "vm/display";
    }

    @Override
    protected GuacamoleTunnel doConnect(HttpServletRequest request) throws GuacamoleException
        // Create our configuration
        GuacamoleConfiguration config = new GuacamoleConfiguration();
        config.setProtocol("rdp");
        config.setParameter("hostname", "localhost");
```

```
        config.setParameter("port", this.port);
        config.setParameter("password", "potato");
        // Connect to guacd - everything is hard-coded here.
        GuacamoleSocket socket = new ConfiguredGuacamoleSocket(
                new InetGuacamoleSocket("localhost", 4822),
                config
        );

        // Return a new tunnel which uses the connected socket
        return new SimpleGuacamoleTunnel(socket);

    }

}
```

Frontend

Para mostrar una máquina virtual en el frontend se necesita usar guacamole-common-js y un div con un id="display". Después se ejecuta un script donde crea un cliente que se comunicará en la url '/tunnel' y se connectará con la función connect(), que llamará al doConnect visto en el apartado anterior. Por último hay código para capturar los eventos del ratón y del teclado.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <!-- Guacamole -->
    <script type="text/javascript"
            src="guacamole-common-js/all.min.js"></script>

    <!-- Display -->
    <div id="display"></div>
    <!-- Init -->
    <script type="text/javascript" th:inline="javascript"> /* <![CDATA[ */
    // Get display div from document
    var display = document.getElementById("display");
    // Instantiate client, using an HTTP tunnel for communications.
    var port = /*[[${port}]]*/ "";
    var guac = new Guacamole.Client(
        new Guacamole.HTTPTunnel("tunnel")
    );
```

```
    // Add client to display div
    display.appendChild(guac.getDisplay().getElement());

    // Error handler
    guac.onerror = function(error) {
        alert(error);
    };

    // Connect
    guac.connect();

    // Disconnect on close
    window.onunload = function() {
        guac.disconnect();
    }

    // Mouse
    var mouse = new Guacamole.Mouse(guac.getDisplay().getElement());

    mouse.onmousedown =
        mouse.onmouseup   =
            mouse.onmousemove = function(mouseState) {
                guac.sendMouseState(mouseState);
            };

    // Keyboard
    var keyboard = new Guacamole.Keyboard(document);

    keyboard.onkeydown = function (keysym) {
        guac.sendKeyEvent(1, keysym);
    };

    keyboard.onkeyup = function (keysym) {
        guac.sendKeyEvent(0, keysym);
    };

    /* ]]> */ </script>
</body>
</html>
```

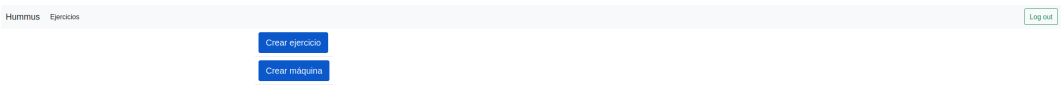
Capítulo 3

Simulación de un caso

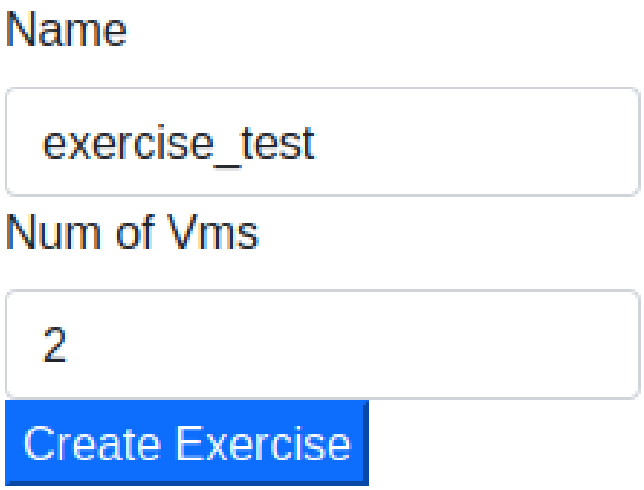
En este capítulo haremos el recorrido que hace el usuario administrador para añadir máquinas virtuales y ejercicios a la aplicación. También se hará un recorrido de un usuario normal hasta abrir un ejercicio y las máquinas virtuales asignadas al ejercicio. Para este recorrido se asumirá que ya existen un usuario normal y otro administrador, además de dos máquinas virtuales instaladas con nombre test y test2.

3.1. Administrador

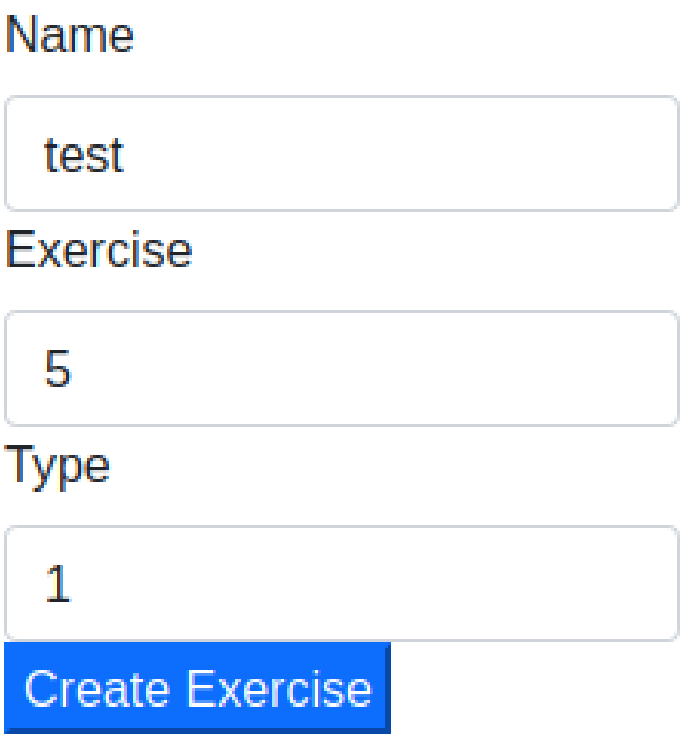
Lo primero que haremos será iniciar sesión con el usuario administrador, seremos redirigidos a la página de ejercicios, pero iremos a la url /admin":



Después crearemos un ejercicio rellenando el formulario que se muestra a continuación:



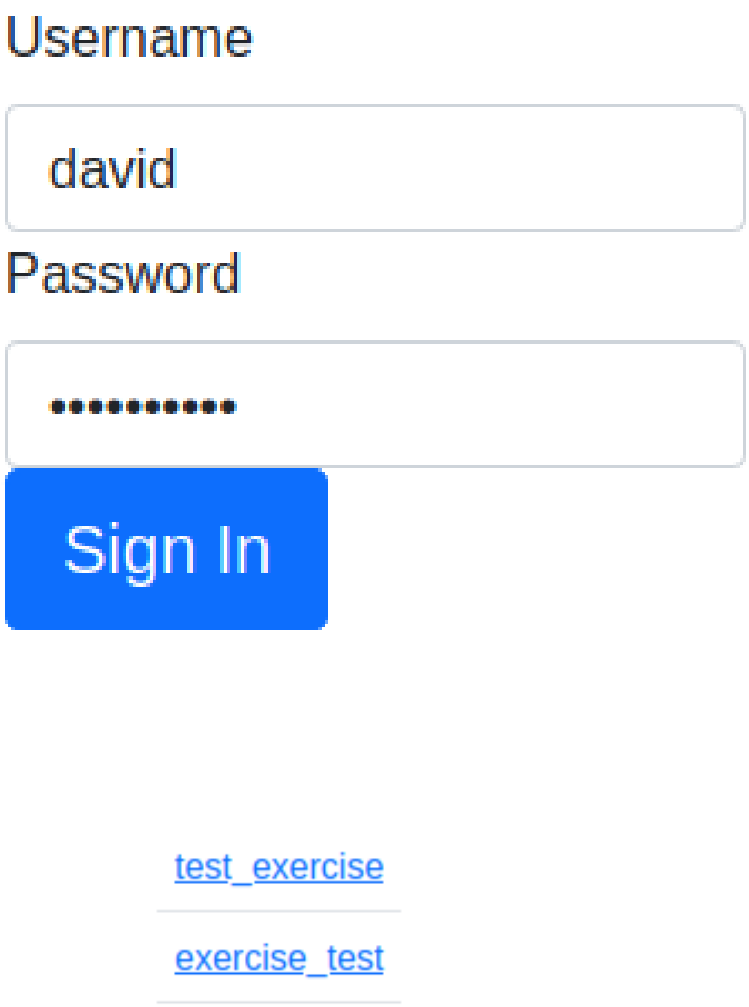
Para terminar añadiremos las máquinas virtuales necesarias para el ejercicio rellenando el siguiente formulario:



Añadiremos una nueva máquina virtual pero solo cambiando el valor de type de 1 a 2. La id del ejercicio en este caso es 5 porque cree ejercicos de prueba con anteroridad durante el desarrollo del trabajo. Para saber el id del ejercició hay que buscarlo en la base de datos.

3.2. Usuario

Iniciaremos sesión y el usuario será redirigido a la página con los ejercicios:

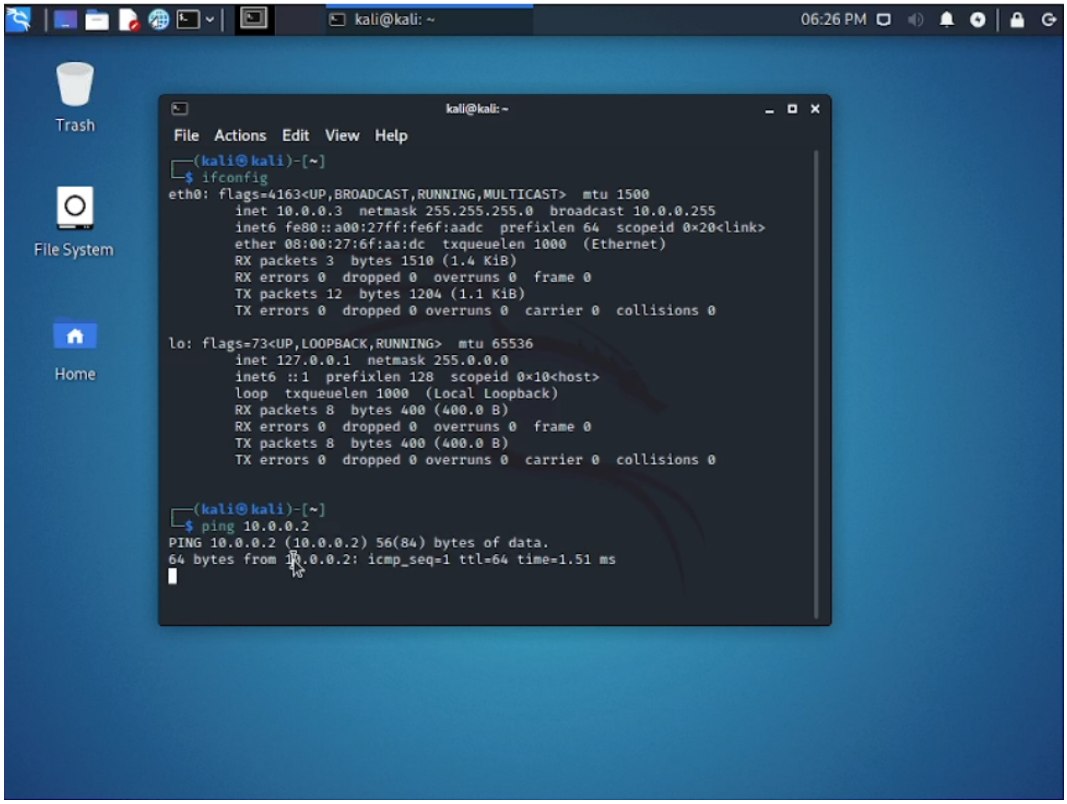


Entraremos en el ejercicio que hemos creado y nos aparecerán los links a las máquinas virtuales:

test

test2

Abriremos las dos en dos pestañas distintas y en una de las dos que hemos abierto abriremos la consola y haremos un ping a la otra para ver que están conectadas:



Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

Hemos visto con este trabajo que es posible integrar estas tecnologías para crear una aplicación web donde se pueden ofrecer máquinas virtuales a usuarios e incluso conectarlas entre sí. Puede ser muy útil para enseñar ciberseguridad a través de problemas a resolver o bien problemas de DevOps donde configurar un sistema para ciertas tecnologías.

4.2. Trabajo futuro

A pesar de todo lo visto creo que aún hay trabajo por hacer para mejorar y perfeccionar el producto, pues en el estado actual no podría salir al mercado. Procedo a listar a continuación algunas mejoras que creo que se pueden hacer:

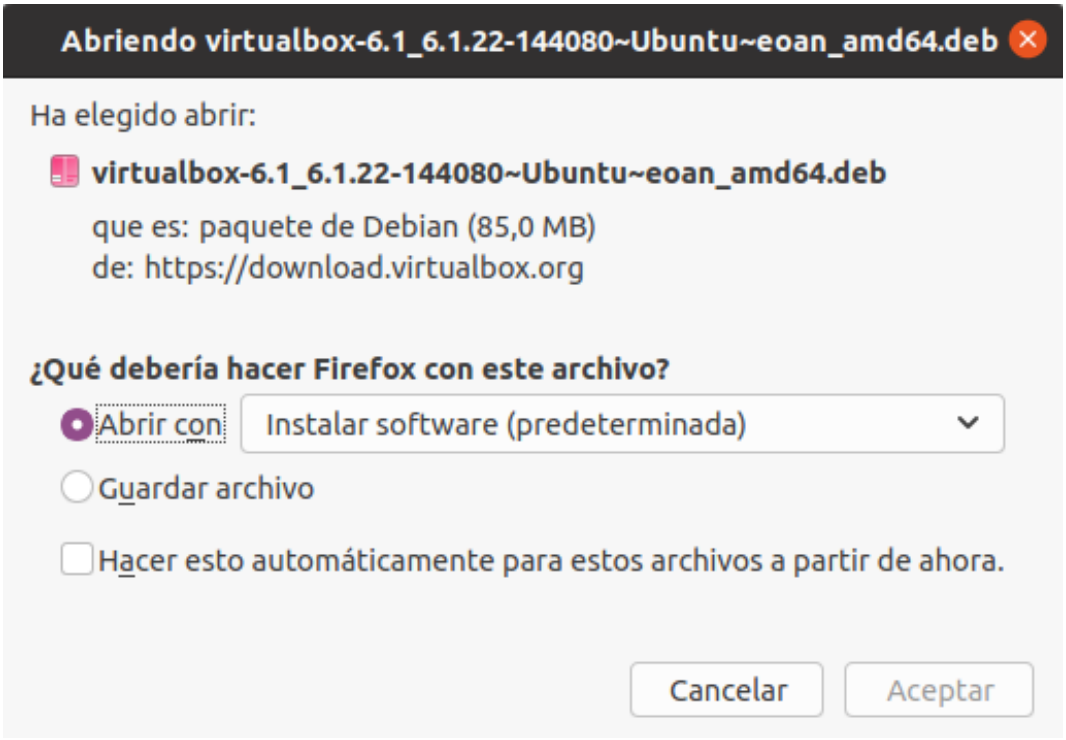
- **Mejorar el frontend y UX:** Este trabajo se ha centrado mucho en la interacción de Spring, el sdk de Virtualbox y Guacamole en una sola aplicación y se puede hacer una interfaz mucho más bonita y mejorar la experiencia de usuario.
- **Crear un set de máquinas preparadas:** Se pueden preparar una série de máquinas virtuales con vulnerabilidades a explotar como por ejemplo versiones desactualizadas, sistemas operativos antiguos o sistemas de seguridad desactivados para poder atacar el sistema.
- **Mejorar el sistema de añadir máquinas virtuales:** Actualmente el procedimiento para añadir una nueva máquina virtual es muy manual y a la que crezca un poco la aplicación a nivel de usuarios lo haría inusable.
- **Mejorar la interacción con el sdk de Virtualbox:** En muchos casos si no se tiene cuidado al cerrar una pestaña o al pulsar el botón de retroceder del navegador es muy fácil que una máquina virtual pete y salga un error. Se debería tener en cuenta esos casos específicos.

Capítulo 5

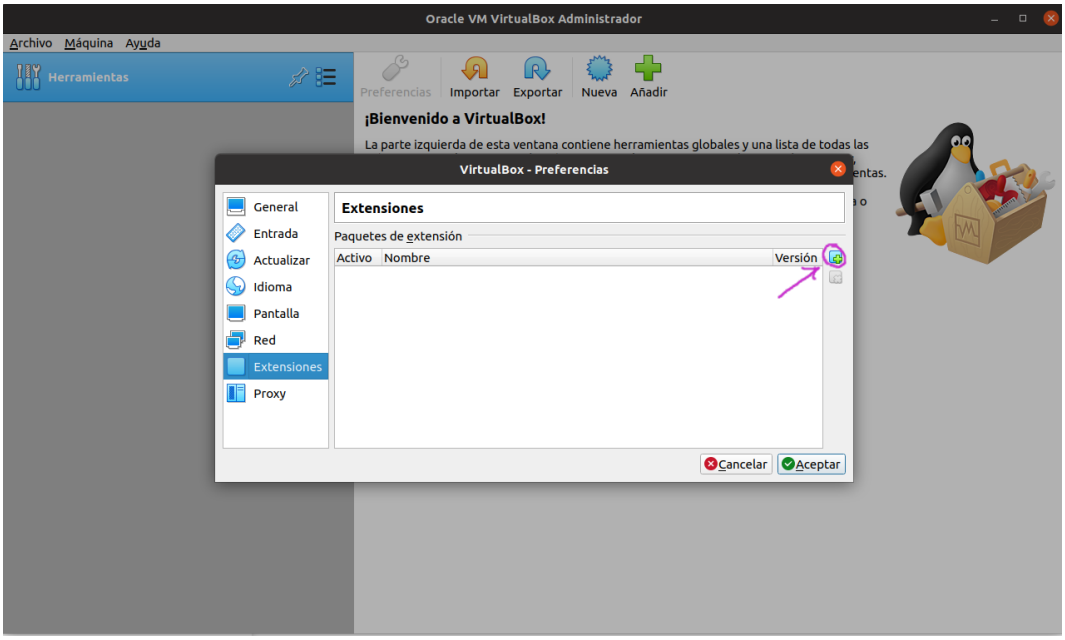
Apéndice: Manual Técnico

5.1. Virtualbox

Lo primero de todo será instalar virtualbox para poder virtualizar las máquinas. Para ello iremos a https://www.virtualbox.org/wiki/Linux_Downloads (En mi caso pongo el link directo a las distribuciones Linux) y descargo la version compatible con Ubuntu 20. Se nos descargará un archivo .deb, que solo con clicar dos veces se nos abrirá el gestor de software de ubuntu e instalará Virtualbox en nuestro ordenador.



Como se comentó en el apartado de introducción para poder conectarse a las máquinas virtuales via RDP hay que instalar también el Virtualbox Extension Pack. Para ello iremos a https://www.virtualbox.org/wiki/Downloads y clicaremos el link en el apartado de VirtualBox 6.1.22 Oracle VM VirtualBox Extension Pack. Se nos descargará un archivo vbox-extpack y lo guardamos en una carpeta. Después desde virtualbox iremos a Archivo ->Preferencias ->Extensiones y le daremos al botón con un signo de suma de color verde. Una vez cliquemos se abrirá el explorador de Ubuntu y buscaremos y seleccionaremos el archivo de extensión vbox-extpack que acabamos de descargar. Seguiremos las instrucciones y ya lo tendremos listo.



5.2. Guacamole

Para instalar Guacamole[3] deberemos ir a https://guacamole.apache.org/releases/ donde están todas las versiones de guacamole. Descargaremos la última versión (al momento de hacer el proyecto era la 1.2.0 pero posteriormente se actualizó a la 1.3.0) y tendremos que seleccionar el archivo guacamole-server-1.3.0.tar.gz.

Antes de hacer nada hay que instalar ciertas dependencias según nos dicen en el manual de apache guacamole: https://guacamole.apache.org/doc/gug/installing-guacamole.html

Podemos instalar las dependencias necesarias ejecutando este comando[5]:

```
$ sudo apt install -y gcc vim curl wget g++ libcairo2-dev libjpeg-turbo8-dev
libpng-dev libtool-bin libossp-uuid-dev libavcodec-dev libavutil-dev
libswscale-dev build-essential libpango1.0-dev libssh2-1-dev libvncserver-dev
libtelnet-dev libssl-dev libvorbis-dev libwebp-dev
```

Una vez instaladas las dependencias vamos a descomprimir el archivo que hemos descargado anteriormente con la comanda:

```
$ tar -xzf guacamole-server-1.3.0.tar.gz
```

Las dependencias son las siguientes:

Library name	Features
Cairo	Cairo is used by libguac for graphics rendering. Guacamole cannot function without Cairo installed. **Debian / Ubuntu package**: libcairo2-dev **Fedora / CentOS / RHEL package**: cairo-devel
libjpeg-turbo	libjpeg-turbo is used by libguac to provide JPEG support. Guacamole will not build without this library present: **Debian package**: libjpeg62-turbo-dev **Ubuntu package**: libjpeg-turbo8-dev **Fedora / CentOS / RHEL package**: libjpeg-turbo-devel If libjpeg-turbo is unavailable on your platform, and you do not wish to build it from source, libjpeg will work as well, though it will not be quite as fast: **Debian / Ubuntu package**: libjpeg62-dev **Fedora / CentOS / RHEL package**: libjpeg-devel
libpng	libpng is used by libguac to write PNG images, the core image type used by the Guacamole protocol. Guacamole cannot function without libpng. **Debian / Ubuntu package**: libpng12-dev **Fedora / CentOS / RHEL package**: libpng-devel
libtool	libtool is used during the build process. libtool creates compiled libraries needed for Guacamole. **Debian / Ubuntu package**: libtool-bin **Fedora / CentOS / RHEL package**: libtool
OSSP UUID	OSSP UUID is used by libguac to assign unique IDs to each Guacamole connection. These unique IDs are the basis for connection sharing support. **Debian / Ubuntu package**: libossp-uuid-dev **Fedora / CentOS / RHEL package**: uuid-devel
FreeRDP	FreeRDP 2.0.0 or later is required for RDP support. If you do not wish to build RDP support, this library is not needed. **Debian / Ubuntu package**: freerdp2-dev **Fedora / CentOS / RHEL package**: freerdp-devel

Y entraremos via terminal a la carpeta que hemos descomprimido para ejecutar el siguiente script:

```
$ ./configure --with-init-dir=/etc/init.d
```

Este script mirará si tenemos instaladas las dependencias mencionadas anteriormente. Si todo es correcto ejecutaremos el comando make. Después del make haremos make install y ldconfig. Si no hemos tenido ningún error estará todo instalado pero tendremos que iniciar el servicio de guacd con la comanda a continuación. En mi caso tuve que reiniciar el ordenador para poder correr el daemon guacd.

```
$ sudo service guacd start
```

5.3. Java

Como comentamos en la introducción por temas de dependencias con el sdk de virtualbox deberemos instalar en nuestro sistema la versión de Java 8. En vez de instalar la versión de Oracle utilizaremos AdoptOpenJDK, que es open-source y la licencia es gratuita. Para instalarlo y hacerlo de forma fácil utilizaremos sdkman. Las instrucciones para instalarlo se pueden encontrar en https://sdkman.io/install.Una vez instalado ejecutaremos la comanda:

```
$ sdk list java
```

Y cogeremos la versión de java deseada, en este caso la versión con id 8.0.292.j9-adpt y para instalar ejecutaremos lo siguiente

```
$ sdk install 8.0.292.j9-adpt
```

5.4. Postgresql

Lo primero será instalar Postgres[2], para ello utilizaremos el siguiente comando:

```
$ sudo apt install postgresql postgresql-contrib
```

Ahora crearemos un usuario aparte que sea el responsable de usar las bases de datos.

```
$ sudo -u postgres createuser --interactive
```

y nos aparecerá los siguientes mensajes:

```
david@david:~$ sudo -u postgres createuser --interactive
[sudo] contraseña para david:
Ingrese el nombre del rol a agregar: sora1
¿Será el nuevo rol un superusuario? (s/n) S^C
david@david:~$ sudo -u postgres createuser --interactive
Ingrese el nombre del rol a agregar: sora
¿Será el nuevo rol un superusuario? (s/n) s
```

Para agregarle a la cuenta una contraseña ejecutaremos lo siguiente:

```
$ sudo -u user psql
```

y en la consola que se nos abre ejecutaremos:

```
$ ALTER USER postgres PASSWORD 'myPassword';
```

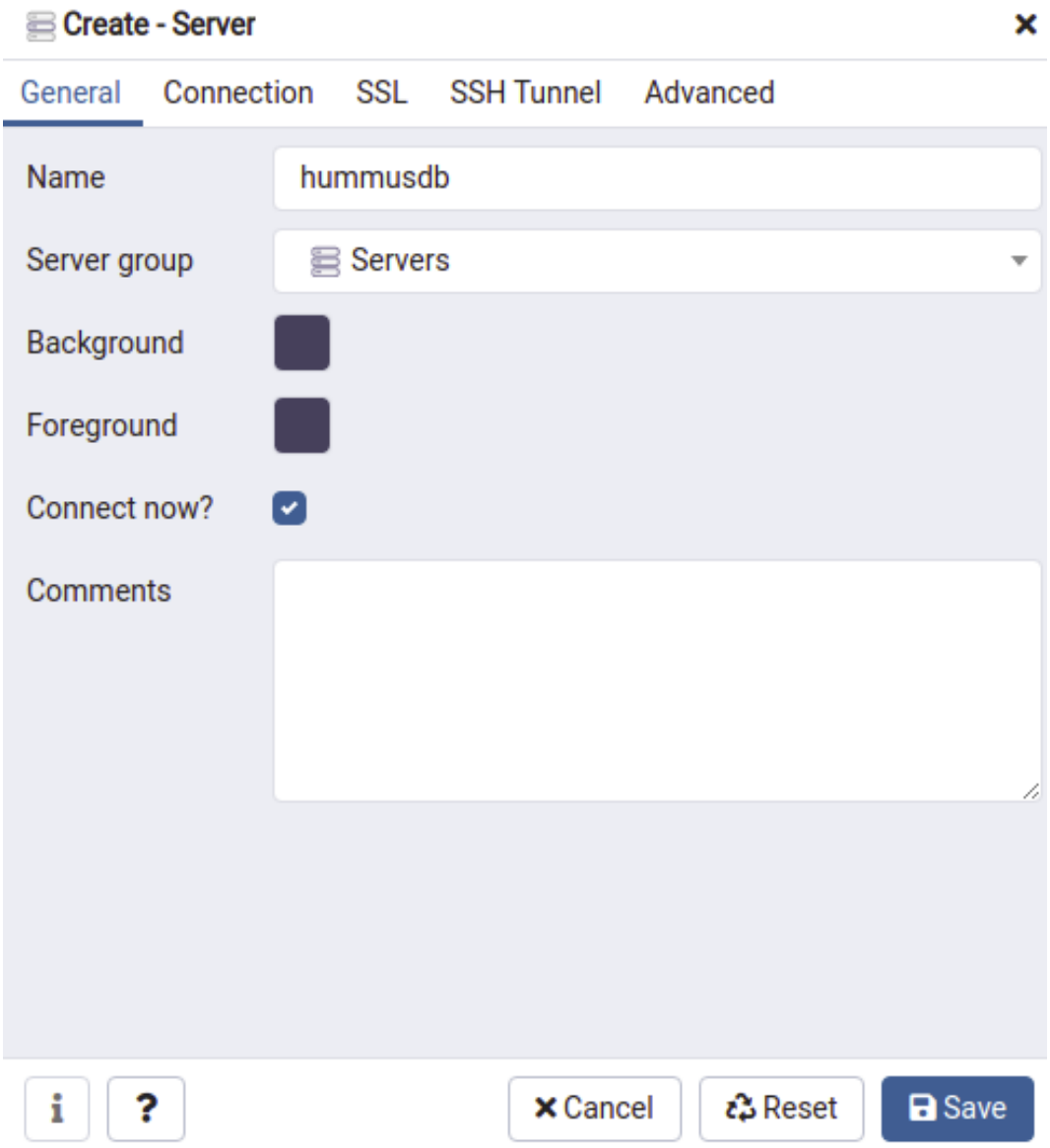
Y crearemos la base de datos que utilizaremos en el proyecto con el usuario que hemos creado anteriormente:

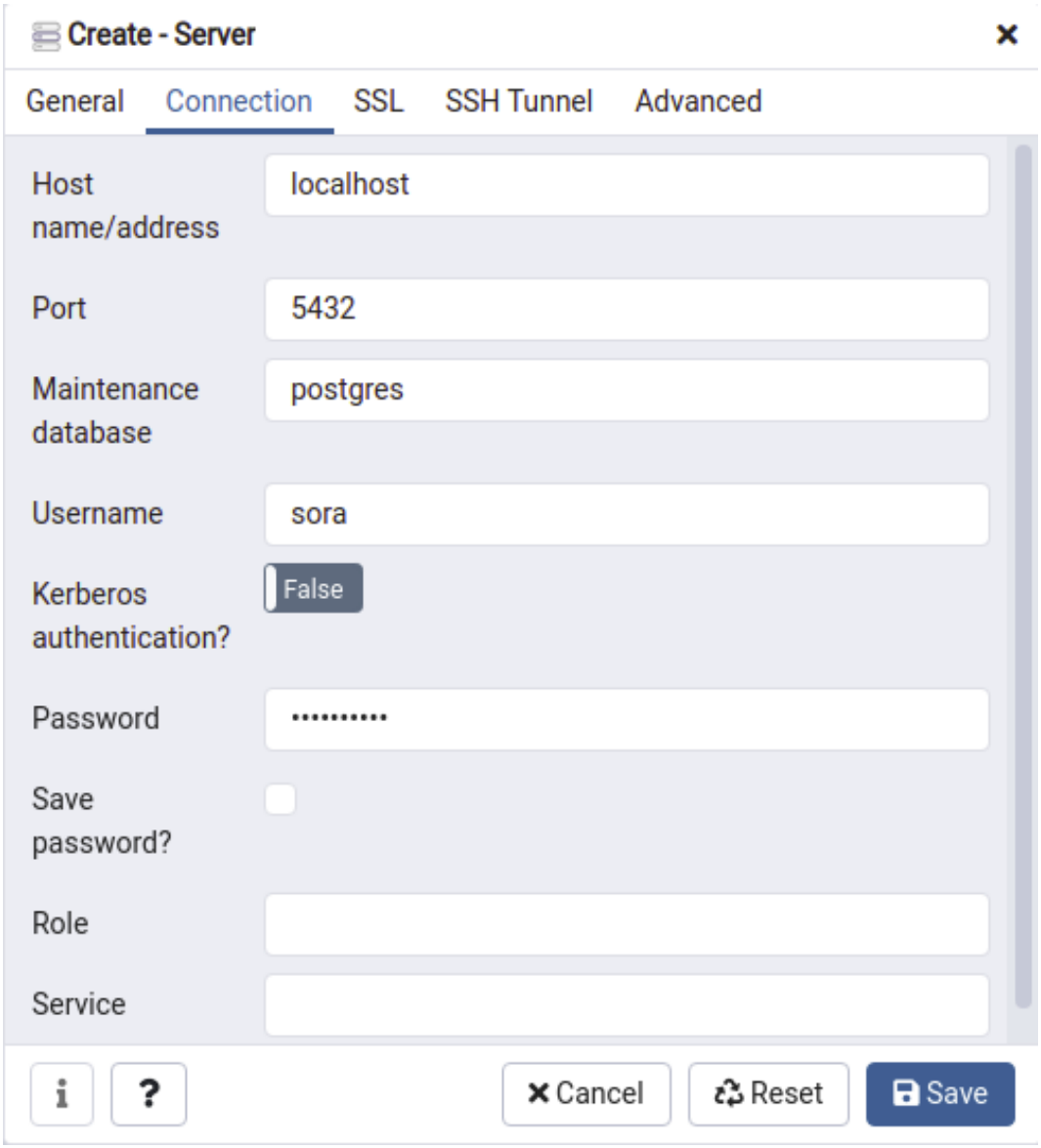
```
$ sudo -u user createdb hummusdb
```

Para conectarnos a la base de datos y poder hacer queries yo he decidido usar PgAdmin4. Para ello utilizaremos los siguientes comandos.

```
# Install the public key for the repository (if not done previously):
$ sudo curl https://www.pgadmin.org/static/packages_pgadmin_org.pub | sudo apt-key add

# Create the repository configuration file:
$ sudo sh -c 'echo "deb https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release

#
# Install pgAdmin
#

# Install for both desktop and web modes:
$ sudo apt install pgadmin4
```

Y para conectarnos a la base de datos que creamos anteriormente tendremos que abrir PgAdmin, darle a Object-create database y entrar nuestros datos como se muestra a continuación:





Ahora debemos poner los credenciales del usuario del base de datos en Spring. Para conectarnos a nuestra base de datos tendremos que abrir el archivo application.properties y poner nuestros credenciales, además de unos campos extra como se ve en la siguiente imagen:

```
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.datasource.url=jdbc:postgresql://localhost:5432/hummusdb
spring.datasource.username=sora
spring.datasource.password=holaquetal
spring.jpa.hibernate.ddl-auto= update
```

5.5. Ejecutar proyecto

Debido a la libreria externa de virutalbox, Maven no encuentra la librería y la solución propuesta es que desde IntelliJ se añada la librería al classpath desde File->Project Structure->SDK y en el signo + en la pestaña classpath añadir el path de la libreria que se encuentra dentro del proyecto en src/main/resources/libs/vboxjws.jar

Bibliografía

[1] Bose, M. Virtualbox vs vmware. https://www.nakivo.com/blog/vmware-vs-virtual-box-comprehensive-comparison/.

[2] Drake, M. Cómo instalar y utilizar postgresql en ubuntu 20.04. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-20-04-es.

[3] Guacamole. Apache guacamole. https://guacamole.apache.org/.

[4] jazzdd86. Phpvirtualbox. https://github.com/jazzdd86/phpVirtualbox.

[5] John, K. Guacamole dependencies. https://computingforgeeks.com/install-and-use-guacamole-on-ubuntu/.

[6] Klinchin, M. Compare rdp vs vnc in simple language. https://www.xtontech.com/blog/rdp-vs-vnc-access/.

[7] Shovon, S. How to access virtualbox 6 vms remotely. https://linuxhint.com/access_virtualbox6_vms_remotely/.

[8] Virtualbox. Licensing faq. https://www.virtualbox.org/wiki/Licensing_FAQ.

[9] VMWare. Vmware store. https://store.vmware.com/sstore?Action=home&SiteID=vmwde&Locale=es_ES&Currency=EUR&id=HomeOffersPage.