

# Frankfurt University of Applied Sciences

– Faculty of Computer Science and Engineering –

## Erweiterung einer webbasierten Remote-Desktop-Anwendung um Audioübertragung und Druckfunktionalität

Abschlussarbeit zur Erlangung des akademischen Grades  
Master of Science (M.Sc.)

vorgelegt am 12. November 2024 von

**Alexander Atanassov**

Matrikelnummer: 1221846

Referent : Prof. Dr. Christian Baun  
Korreferent : Johannes Bouché

# Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Frankfurt, 12. November 2024

A. Atanassov  
**Alexander Atanassov**

# Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Ich bedanke mich bei meinem Betreuer, Prof. Dr. Baun, für seine wertvolle Unterstützung und die Möglichkeit, diese Arbeit zu schreiben. Ebenso danke ich Herrn Bouché, meinem Zweitprüfer, für die konstruktive Kritik und die hilfreichen Hinweise.

Ein herzlicher Dank geht auch an meine Familie, die mich während der gesamten Arbeit und Studium mit viel Geduld und Verständnis unterstützt haben. Ohne diese Unterstützung wäre diese Arbeit nicht möglich gewesen.

# Kurzfassung

Diese Arbeit befasst sich mit der Entwicklung einer Web-Anwendung, die es ermöglicht Computer-Desktops fernzusteuern und dabei sowohl RDP-als auch VNC-Verbindungen unterstützt. Die Übertragung der Daten bis zu dem Webbrowser basiert auf das Guacamole-Protokoll von Apache. Zusätzlich wurden Methoden entwickelt um Audio zu übertragen und das Drucken von Dokumenten zu ermöglichen. Zu dem Thema wurden in der Literatur nur Arbeiten gefunden, die sich mit der Einrichtung von Guacamole und der Integration der Audiowiedergabe beschäftigen, jedoch keine Ansätze zur Druckfunktion bieten. Für die Audioübertragung bei RDP-Verbindungen wurde das RDP-Audio Konzept sowohl für die Wiedergabe als auch für die Aufnahme von Audiodaten verwendet. Bei VNC-Verbindungen wurde der PulseAudio-Server auf den Zielgeräten benutzt, die Audiodaten zu senden und zu empfangen. Für die Druckfunktion wurden Lösungen entwickelt, wo der Benutzer der Anwendung den Druckauftrag an einen virtuellen Drucker sendet, welcher eine PDF-Datei daraus generiert, die im Browser heruntergeladen werden kann. Bei RDP-Verbindungen wurde dafür GhostScript verwendet, während bei VNC-Verbindungen ein CUPS-Server mit CUPS-PDF Druckern zum Einsatz kam. Die Ergebnisse dieser Arbeit zeigen, dass das Drucken von Dokumente über PDF funktioniert, allerdings erfordert dieser Ansatz bei größeren Aufträge viel Zeit. Die Audioübertragung funktioniert ebenfalls für alle Verbindungsarten. Bei VNC-Verbindungen gibt es jedoch Signalisierung, die erkennt wann der Benutzer das Mikrofon nutzt.

# Abstract

This work focuses on the development of a web-application that allow users to control computer-desktops remotely, supporting RDP and VNC connections. Data transmission to the web browser is based on Apache's Guacamole protocol. Additionally, solutions have been developed to enable audio transmission and a printing option. In the literature, only works about Guacamole setup and audio integration could be found, but no approaches adressing printing functionality. For audio transmission over RDP connections, the RDP-audio feature was used for both playback and recording of audio data. For VNC connections, the PulseAudio server on the remote devices was used to stream and receive audio data. For the printing feature, solutions have been developed where the print jobs are send to a virtual printer that converts them to PDF and send them to the browser where the user can download them. For RDP connections GhostScript have been used and for VNC connections a CUPS server with CUPS-PDF printers. The results of this work show that printing documents as PDFs works, but large documents require significant time. Audio transmission in both direction works for all connection types. However, for audio recording over VNC, there is no signaling to indicate when the user accesses the microfon.

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>4</b>
1.1	Motivation . . . . .	4
1.2	Das Desktop as a Service Modell . . . . .	5
1.3	Herausforderungen bei der Integrierung von Audio und Drucken . . . . .	6
<b>2</b>	<b>Stand der Technik</b>	<b>7</b>
2.1	Zielsetzung und Kriterien . . . . .	7
2.2	Verwandte Arbeiten . . . . .	8
2.3	Lösungen für eine Druckoption . . . . .	9
2.4	Lösungen für Audioübertragung . . . . .	11
2.5	Verwendete Technologien . . . . .	12
2.5.1	Services . . . . .	12
2.5.1.1	Proxmox . . . . .	13
2.5.1.2	Apache Guacamole . . . . .	14
2.5.1.3	CUPS . . . . .	16
2.5.1.4	PulseAudio . . . . .	16
2.5.1.5	Docker . . . . .	18
2.5.2	Protokolle . . . . .	19
2.5.2.1	Guacamole Protokoll . . . . .	19
2.5.2.2	RDP Protokoll . . . . .	20
2.5.2.3	VNC Protokoll . . . . .	21
2.5.2.4	Internet Printing Protokoll . . . . .	21
2.5.2.5	WebSocket . . . . .	23
2.5.3	Webtechnologien . . . . .	25
2.5.3.1	Node.js und Express . . . . .	25

2.5.3.2	React JS . . . . .	25
2.5.3.3	Verschlüsselung der Verbindungen mit TLS . . . . .	26
<b>3</b>	<b>Design</b>	<b>28</b>
3.1	Design der Basis Remote-Desktop Anwendung . . . . .	28
3.2	Erweiterung um eine Druckoption . . . . .	30
3.3	Erweiterung um Audioübertragung . . . . .	30
3.4	Gesamtarchitektur . . . . .	31
<b>4</b>	<b>Implementierung</b>	<b>33</b>
4.1	Entwicklungsumgebung . . . . .	33
4.2	Implementierung der Basisanwendung . . . . .	35
4.3	Implementierung der Druckeroption . . . . .	41
4.4	Dateitransfer für RDP . . . . .	47
4.5	Implementierung der Audioübertragung . . . . .	48
4.6	Sicherheit . . . . .	55
4.7	Bereitstellung . . . . .	56
<b>5</b>	<b>Bewertung</b>	<b>58</b>
5.1	Ziele und Ergebnisse . . . . .	58
5.2	Bewertung der Methodik . . . . .	59
5.3	Bedeutung der Ergebnisse . . . . .	60
<b>6</b>	<b>Schluss</b>	<b>61</b>
6.1	Zusammenfassung . . . . .	61
6.2	Ausblick . . . . .	62
6.3	Fazit . . . . .	62
<b>A</b>	<b>Programmcode</b>	<b>64</b>
	<b>Literatur</b>	<b>65</b>

# Abbildungsverzeichnis

2.1	Hypervisor Typ 1: direkt auf der Hardware, Typ 2: benötigt Host-OS . . . . .	13
2.2	Architekturdiagramm zur Ausführung von VMs unter Linux . . . . .	14
2.3	Architektur des Apache Guacamole Projekts . . . . .	15
2.4	Überblick von Sources und Sinks unter PulseAudio . . . . .	17
2.5	Design des IPP-Modells . . . . .	22
2.6	Verbindungsaufbau und Nachrichtenaustausch bei WebSocket Verbindungen	24
2.7	TLS-Handshake Sequenz . . . . .	27
3.1	Architektur der Basis-Anwendung. . . . .	29
3.2	Erweiterung für VNC Verbindungen . . . . .	32
4.1	Web-basierte Benutzeroberfläche von Proxmox-VE. . . . .	34
4.2	Web-basierte Benutzeroberfläche von Apache Guacamole . . . . .	36
4.3	Formular für die Herstellung von Verbindungen . . . . .	38
4.4	Linux Mint Verbindung über VNC . . . . .	40
4.5	Windows 10 Verbindung über RDP . . . . .	41
4.6	Guacamole Drucker unter Windows . . . . .	42
4.7	Guacd-Logs beim Drucken mit dem Guacamole-Printer . . . . .	42
4.8	PDF Drucker für einen Beispielbenutzer . . . . .	46
4.9	Guacamole Laufwerk unter RDP . . . . .	48
4.10	Remote Audio unter Windows RDP Server . . . . .	51
4.11	Erkannte Eingabegeräte von PulseAudio . . . . .	52
4.12	Toolbar mit Button für Audioaufnahmen . . . . .	53
4.13	TLS Zertifikat von den Server . . . . .	56
4.14	Container-basierte Ansicht der Anwendung . . . . .	57



# Verzeichnis der Listings

1	Beispiele für Guacamole-Protokoll-Anweisungen . . . . .	20
2	Client-Anfrage zum Wechsel zu WebSocket . . . . .	23
3	Bestätigung vom Server zum Wechsel zu WebSocket . . . . .	24
4	x11vnc.service Konfiguration . . . . .	35
5	Ausführen von einem guacd-Container . . . . .	35
6	Beispiel für einen einfachen guacamole-lite Server . . . . .	37
7	Verbindungsaufbau zu dem Guacamole-Server . . . . .	39
8	Implementierung von Listener für Maus- und Tastatur-Eingabe . . . . .	39
9	Guacd-Logs bei einer RDP-Verbindung . . . . .	41
10	Funktion zum Herunterladen von Dateien über das Guacamole-Protokoll . . . . .	43
11	Konfiguration der ACLs . . . . .	44
12	Konfigurierung des Ablageorts von PDF-Dateien, erzeugt von CUPS-PDF . . . . .	45
13	Hinzufügen von CUPS-PDF Drucker . . . . .	45
14	Erstellung von IPP-Drucker Instanz . . . . .	46
15	Guacd-Log für die Akzeptierten Audioformate . . . . .	49
16	Verbindungsaufbau mit Angabe des Audioformats . . . . .	49
17	Funktion zur Aufnahme von Audio bei RDP-Verbindungen . . . . .	50
18	Erstellung von virtuellen Mikrofon . . . . .	52
19	Audiodaten Empfang über Netcat . . . . .	53
20	Pacat als Unterprozess für Node . . . . .	54
21	Erzeugung von TLS/SSL Zertifikat mit OpenSSL . . . . .	55

# Abkürzungsverzeichnis

**ACL** Access Control Lists.

**AJAX** Asynchronous JavaScript and XML.

**CUPS** Common Unix Printing System.

**DaaS** Desktop-as-a-Service.

**GDI** Graphics Device Interface.

**IPP** Internet Printing Protocol.

**KVM** Kernel-based Virtual Machine.

**LXC** Linux Container.

**PVE** Proxmox Virtual Environment.

**QEMU** Quick Emulator.

**RDP** Remote Desktop Protokoll.

**SaaS** Software as a Service.

**SPICE** Simple Protocol for Independent Computing Environments.

**SSH** Secure Shell.

**SSL** Secure Socket Layer.

**TLS** Transport Layer Security.

**VM** Virtuelle Maschinen.

**VNC** Virtual Network Computing.

**WebRTC** Web Real-Time Communication.

# Glossar

**assistive Technologien** Assistive Technologien sind Hilfsmitteln für Menschen mit Behinderungen um die Benutzung von Computern zu vereinfachen.

**Guacamole** Open-Source-Dienst für Desktopübertragung im Browser.

**guacd** Der Guacamole-Server (guacd) ist ein Hintergrunddienst, der die Guacamole-Verbindungen steuert.

**Hypervisor** Ein Hypervisor ist eine Software, mit der mehrere virtuelle Maschinen auf einer einzigen physischen Maschine ausgeführt werden können.

**Peer-to-Peer** Ein Netzwerkmodell, bei dem jeder Knoten direkt mit anderen Knoten kommunizieren kann, ohne auf einen zentralen Server angewiesen zu sein..

**Remote-Desktop** Der Desktop von einen Computer, der ferngesteuert wird.

**Remote-Desktop-Viewer** Eine Anwendung oder Teil einer Anwendung, über die Computer-Desktops ferngesteuert werden können.

# Kapitel 1

## Einführung

Dieses Kapitel gibt einen Überblick über den Bedarf an flexiblen Arbeitsmodellen, das Desktop-as-a-Service (DaaS)-Konzept und die Herausforderungen bei der Integration von Audio- und Druckfunktionen in webbasierten Remote-Desktop-Lösungen. Es wird erläutert, wie Remote-Arbeit und Lernen durch DaaS ermöglicht wird und warum das SaaS-Modell an seine Grenzen kommt.

### 1.1 Motivation

In der heutigen dynamischen Arbeitswelt gibt es ein wachsendes Bedürfnis nach flexiblen Arbeitsmodellen. Unternehmen und Mitarbeiter schätzen die Möglichkeit, von verschiedenen Standorten aus arbeiten zu können, was zu erhöhter Mobilität und einer besseren Work-Life-Balance führt. Um diese Ziele zu erfüllen wurden verschiedene Konzepte entwickelt. Ein häufig benutztes Konzept sind komplexe interaktive Webanwendungen. Mithilfe von moderne Technologien wie Asynchronous JavaScript and XML (AJAX) [42], Webservice Protokolle und HTML5 lassen sich heutzutage komplizierte Anwendungen auch für Webbrowser implementieren. Damit Firmen solche Anwendungen nicht selbst verwalten müssen, wurde das Software as a Service (SaaS) [6] Konzept entwickelt. Dabei übernehmen Anbieter den Betrieb und Wartung der Software und erhalten dafür Gebühren. Um die Anwendungen zu benutzen wird in der Regel nur ein Gerät mit einen Webbrowser und Internetzugang benötigt.

Webanwendungen stoßen allerdings bei bestimmten Anwendungsfällen an ihre Grenzen. Komplexe Programme, die hohe Rechenleistung, speziellen Hardwarezugriff oder stark von dem Betriebssystem abhängig sind, sind oft nur als Desktop-Anwendungen verfügbar. Zudem existieren viele bereits bestehende Programme nur als Desktop-Varianten und müssten als Webanwendungen neu-konzipiert werden. Um Anwendungen, die speziell für Desktopumgebungen entwickelt wurden, in flexible Arbeitsmodelle zu integrieren, wurde das Konzept des Desktop-as-a-Service (DaaS) entwickelt. Dabei werden Desktop-Anwendungen in der Cloud ausgeführt und die Benutzeroberfläche wird mithilfe verschiedener Protokolle übertragen.

Eine zentrale Herausforderung bei solchen Lösungen ist jedoch die volle Integration von Funktionen, die für den Benutzer wichtig sind um die Anwendungen vollständig zu benutzen. Dazu gehören unter anderem die Audioübertragung und das Drucken von Dokumente. Diese Funktionen sind entscheidend, da ohne sie viele Anwendungen unvollständig wären und die Produktivität beeinträchtigen könnten. Die Audioübertragung ist beispielsweise wichtig für Anwendungen wie Video- oder Musikbearbeitung. Eine Druckfunktionalität ist essenziell für viele Branchen, in denen Dokumente noch auf Papier gedruckt werden müssen.

## 1.2 Das Desktop as a Service Modell

DaaS ist ein Cloud-Computing-Angebot, das virtuelle Desktop-Umgebungen bereitstellt, die von einem Dienstleister gehostet und verwaltet werden. Die Desktops werden in Virtuelle Maschinen (VM)en auf den Rechnern des Anbieters ausgeführt und verwaltet. Die Nutzer können sich von jedem Gerät, das Internetzugriff hat, mit den Desktop verbinden. Die gesamte Rechenleistung und Datenspeicherung erfolgt auf den Servern des Anbieters. Dieses Konzept ist attraktiv für Unternehmen, da es die zentrale Verwaltung von Desktops ermöglicht und die Verwaltung von einzelnen Geräte vor Ort vermeidet, was mit Zeit und Kosten verbunden ist. [32]

Der Zugriff auf den Remote-Desktop erfolgt in der Regel über eine Client-Software die zusätzlich installiert und konfiguriert werden muss. Die Installation und Konfiguration kann je nach Gerät und Betriebssystem unterschiedlich sein. Um den Prozess für den Endbenutzer zu vereinfachen, wurden Methoden entwickelt die Client-Software im Webbrowser auszulagern. Somit muss auf dem Endgerät keine zusätzliche Software installiert werden. Das Gerät braucht nur ein aktuellen Webbrowser und eine Internetverbindung um den Remote-Desktop-Viewer zu benutzen. Außerdem ist diese Lösung plattformunabhängig und der Benutzer kann auf jedes beliebige Gerät auf seine Desktop-Umgebung zugreifen.

Die Covid-19 Pandemie hat die Bedeutung von DaaS und ähnlichen Technologien noch weiter hervorgehoben. Viele Bildungseinrichtungen und Unternehmen waren plötzlich gezwungen auf Remote-Arbeit und -Lernen umstellen um den Betrieb aufrechtzuerhalten und den Lernprozess nicht zu unterbrechen. In diesen Zeitraum wurden viele Studien durchgeführt [31, 27, 34], die Lösungen entworfen haben um Studierenden virtuelle Arbeitsumgebungen zu erschaffen. Es kann eine große Herausforderung für Studierenden sein, die benötigte Software lokal zu installieren. Häufig reicht die Rechenleistung ihrer Geräte nicht aus oder sie haben nicht das passende Betriebssystem. Auch das einrichten von VMs kann für einige Studierende kompliziert sein und bei älteren Geräten zu Problemen führen. Mithilfe von Remote-Desktop-Lösungen können alle Studierenden eine einheitlich eingerichtete Arbeitsumgebung nutzen, die zentral verwaltet wird.

## 1.3 Herausforderungen bei der Integrierung von Audio und Drucken

Die Grundfunktionalitäten eines webbasierten Remote-Desktop-Viewers umfassen die Übertragung der Graphiken, Mausinteraktionen und Tastatureingaben. Diese Basisfunktionen allein reichen vielen Nutzern nicht aus. Sie benötigen häufig zusätzliche Funktionalitäten um ihre Arbeit ausführen zu können. Dazu gehört unter anderem die Wiedergabe von Audio und die Nutzung des Mikrofons, was beispielsweise für Video- oder Audibearbeitungssoftware sehr wichtig ist. Auch das Drucken ist für Benutzer besonders wichtig, die erzeugte Dokumente in ausgedruckter Form benötigen. Die Implementierung für solche Funktionalitäten kann aufgrund von viele Faktoren herausfordernd sein.

Diese Arbeit entstand durch Zusammenarbeit mit den DaaS-DESIGN Projekt [4], das eine webbasierte Remote-Desktop Lösung entwickelt, wo verschiedene Desktop-Anwendungen für Windows und Linux im Webbrowser genutzt werden können. Dafür werden die einzelnen Anwendungen in der Cloud in VMs oder Containern ausgeführt, während die Benutzeroberfläche über Fernsteuerungsprotokolle wie Virtual Network Computing (VNC) oder Remote Desktop Protokoll (RDP) übertragen wird. Da der Browser nicht direkt über VNC und RDP kommunizieren kann, wird das Guacamole-Protokoll [25] als Vermittler angewendet. Die Übertragung von Audio und Druckaufträge ist jedoch herausfordernd, besonders für webbasierte Lösungen. Das VNC-Protokoll unterstützt weder Audioübertragung noch einer Druckfunktion. Das bedeutet, dass diese Daten über zusätzliche Verbindungen übertragen werden müssen. Dabei können Probleme wie Latenz, Datenverlust oder Sicherheitslücken entstehen. RDP unterstützt zwar diese Funktionalitäten, deren Integrierung im Webbrowser kann dennoch herausfordernd sein. Es existieren bereits Lösungen für desktopbasierte Remote-Desktop-Viewer, wie der eingebaute Viewer für Windows Pro oder RealVNC, die über diese Funktionalitäten bereits verfügen. Sie haben aber ein besseren Zugriff auf Hardware-Ressourcen als der Browser und können auch direkt über die Fernsteuerungsprotokolle kommunizieren. Zudem sind sie oftmals kostenpflichtig. Für browserbasierte Lösungen besteht weiterhin Verbesserungsbedarf, insbesondere bei VNC-Verbindungen. Ziel dieser Arbeit ist es, Lösungen für die Audioübertragung und Druckoptionen zu entwickeln. Diese Lösungen sollen in das DaaS-DESIGN Projekt integriert werden, um den Nutzern alle wichtigen Funktionalitäten für eine gute Benutzererfahrung zu bieten.

# Kapitel 2

## Stand der Technik

Dieses Kapitel beginnt mit einer Beschreibung der Zielsetzung des Projekts und den Kriterien, die die zu implementierenden Lösung erfüllen muss. Darauf folgt eine Literaturrecherche nach Verwandte Arbeiten, die sich mit Remote-Desktop Lösungen, basierend auf Apache Guacamole, beschäftigen. Als nächstes wird nach nach mögliche Lösungen für Audioübertragung und Druckoptionen untersucht. Zum Schluss wird eine Beschreibung der Technologien gegeben, die für Umsetzung des Projekts benutzt wurden.

### 2.1 Zielsetzung und Kriterien

Ziel dieser Arbeit ist die Entwicklung einer webbasierten Remote-Desktop-Anwendung mit integrierten Audio- und Druckerfunktionen. Die Anwendung soll es ermöglichen, sowohl Windows- als auch Linux-Desktops direkt im Webbrowser darzustellen und zu benutzen. Es soll für den Endnutzer so einfach wie möglich gemacht werden. Das heißt er soll nicht aufgefordert werden etwas bei sich lokal zu installieren oder Konfigurationen durchzuführen. Alles was er benötigen soll ist ein Gerät mit Internetzugriff und ein aktuellen Webbrowser, der HTML5 unterstützt. Außerdem soll das Betriebssystem des Geräts keine Rolle spielen um die Anwendung zu benutzen. Um den Erfolg dieser Anwendung sicherzustellen, wurden Kriterien definiert, die erfüllt werden müssen. Diese bilden den Rahmen für das Design und Implementierung des Projekts und dienen als Maßstab für die Bewertung der Ergebnisse.

Das erste Kriterium ist das Projekt auf Apache Guacamole (Siehe 2.5.1.2) zu basieren. Das bedeutet, dass der Guacamole Server (guacd) als Vermittler zwischen den Remote-Desktop und der Web-Anwendung dienen. Es existieren zwar auch andere Alternativen, einen Desktop im Browser darzustellen, allerdings sollen die entwickelten Lösungen mit dem DESIGN Projekt [4], der auf Guacamole basiert, kompatibel sein. Als weiteres soll eine einfache Benutzeroberfläche implementiert werden. Der Nutzer soll in der Lage sein sich mit eins seiner Desktops zu verbinden und ihn zu steuern. Die Verbindungsdaten können durch ein einfaches Formular angegeben werden. Eine Registrierung oder Anmeldung ist für dieses Projekt nicht erforderlich, da der Fokus auf der Implementierung des Remote-Desktop-Viewers mit den zusätzlichen Funktionalitäten liegt. Die Hauptkriterien des View-

ers sind den Desktop im Browser anzuzeigen und Mausinteraktionen sowie Tastatureingaben weiterzuleiten. Es müssen sowohl RDP als auch VNC (Siehe 2.5.2.2 und 2.5.2.3) Verbindungen unterstützt werden, um eine breite Kompatibilität mit verschiedenen Betriebssystemen und Desktop-Umgebungen zu gewährleisten. Als nächstes muss eine Druckeroption entwickelt werden. Diese soll den Nutzer ermöglichen ein beliebiges Dokument aus der Remote-Desktop-Umgebung mit seinen lokalen Drucker auszudrucken. Grundsätzlich gibt es hier zwei Möglichkeiten. Die erste Möglichkeit ist den Druckauftrag direkt an den lokalen Drucker zu senden. Die Zweite ist eine PDF Datei aus dem Auftrag zu erzeugen, diese an den Web-Client zu schicken, sie im Browser herunterzuladen und dann lokal auszudrucken. Beide Optionen würden die Anforderung erfüllen. Wichtig dabei ist, dass der Nutzer keine zusätzlichen Einstellungen machen soll. Er soll nur den Browser öffnen, sich mit den Desktop verbinden und auf Drucken klicken. Die Kriterien für die Audioübertragung sind die Audioausgabe vom Remote-Desktop im Browser abzuspielen zu können und die Verwendung vom Mikrofon zu ermöglichen. Dabei ist es ebenfalls wichtig, dass der Benutzer keine zusätzliche lokale Änderungen oder Installationen benötigen soll. Die Audioübertragung soll synchron zum Bild sein und die Latenz soll möglichst gering sein. Das gilt sowohl für die Audioausgabe als auch für das Mikrofon. Die Anwendung soll zusätzlich minimale Sicherheitsanforderungen erfüllen. Das bedeutet, dass keine unsichere Verbindungen nach außen geöffnet werden sollen. Sensible Daten wie Passwörter, IP-Adressen sollen nicht unverschlüsselt gesendet werden. Das letzte Kriterium ist die Audio- und Druck-Funktionalitäten wiederverwendbar zu machen. Wie bereits beschrieben, ist das Ziel dieses Projekts, die implementierten Lösungen in das DESIGN Projekt zu integrieren. Aus diesem Grund sollen die Implementierungen entweder als ganze Komponenten oder als einzelnen Klassen zu Verfügung gestellt werden.

## 2.2 Verwandte Arbeiten

Webbasierte Remote-Desktop-Viewer sind kein neues Thema sondern existieren schon länger. Es gibt zahlreiche Projekte, die solche Lösungen implementieren. Viele davon sind kommerziell, wie TeamViewer [59] oder AnyDesk [2]. Es existieren jedoch auch kostenlose Alternativen, wie beispielsweise Chrome Remote Desktop [29], Kasm Workspaces [35], ThinVNC [52] oder Apache Guacamole [20]. Da die Implementierung dieses Projekts auf Apache Guacamole basiert, werden in dieser Untersuchung nur Studien betrachtet, die Lösungen auf Basis von Guacamole entwickeln.

Dazu gibt es einiges an Forschung. Viele Forschungsprojekte wurden während der Covid-19 Pandemie durchgeführt um Schüler und Studenten eine Arbeitsumgebung anzubieten, ohne etwas auf deren Computer installieren zu müssen. I. Hassan [31] entwickelte mithilfe von Apache Guacamole ein System um Studierenden eine vorkonfigurierte Linux-Umgebung für einen Cybersicherheitskurs anzubieten. Dafür wurden Docker und Linuxcontainer mit Ubuntu erstellt. Der Guacamole-Server (guacd) wurde als Vermittler verwendet. Für das Projekt wurde keine eigene Webanwendung implementiert, sondern wurde der offizielle Client des Guacamole-Projekts verwendet. Als Kommunikationsprotokoll wurde Secure Shell (SSH) verwendet. Studierenden nutzten das System während des Semesters für deren Übungen



und bewerteten es im Anschluss. Die Mehrheit war zufrieden mit der Lösung. In einer anderen Studie von S. García et al. [27], die ebenfalls während der Covid-19 Pandemie durchgeführt wurde, wurde ein System entwickelt um Studierenden Labor Übungen von Zuhause anzubieten. Dazu wurden die Labor-Computer sowie VMs verwendet. Guacd und Guacamole-Client wurden als Vermittler und als Web-Client benutzt. Zusätzlich wurde ein weiteres Verwaltungs-Tool entwickelt um Lehrenden den Zugriff und die Verwaltung von den Remote-Maschinen zu ermöglichen. In ein weiteres ähnliches Projekt während der Pandemie wurde eine Arbeitsumgebung für Studierenden der "EUROCONTROL Institute of Air Navigation Services" erschaffen. Ähnlich wie bei den anderen Studien verwendeten M. Janicik et al. [34] ebenfalls den Guacamole-Server und Guacamole-Client. Als Verbindungstyp wurde VNC verwendet. K. Swargiary [56] berichtet ebenfalls über ein ähnliches System, das für Online-Kurse benutzt wurde, um Studierende Fernzugriff auf Desktop-Anwendungen mithilfe von Guacamole zu ermöglichen ohne dafür etwas installieren zu müssen.

Die bisher beschriebenen Projekte hatten das Ziel, den allgemeinen Zugriff auf Remote-Desktops bereitzustellen. Über Sound, Drucken, und andere zusätzliche Funktionalitäten wurde dort nichts erwähnt. Es existieren nur wenige Arbeiten die sich damit beschäftigen. Einige wurden dennoch gefunden, wie z.B. eine Studie von D. Mulfari et al. (2013) [40], in der ein System entwickelt wurde, das Menschen mit Behinderungen ein nutzerfreundliches cloud-basiertes Desktop bietet. Das Ziel des Systems ist Desktops anzubieten wo alle benötigten assistive Technologien drauf installiert sind, damit die Benutzer das nicht selber lokal installieren und konfigurieren müssen. Für das Experiment wurden virtuelle Maschinen in VirtualBox erstellt. Um sie zu verwalten wurde CLEVER verwendet. Guacamole wurde als Gateway zwischen den VMs und der Web-Anwendung verwendet und als Kommunikationsprotokoll wurde VNC verwendet. Bezüglich der Audiofunktion stieß das Team während des Experiments auf das Problem, dass der "Screen Reader" nicht funktionierte, da VNC keine Audioumleitung unterstützt. Um das Problem zu lösen wurde ein Stream-Server aufgesetzt, der das komprimierte Audio von den VMs an der Web-Anwendung schickte. Der Performance-Test zeigte eine durchschnittliche Audioverzögerung von 1384 ms. Das Team stufte damit das Audio-Feature als unbrauchbar. In einer weiteren Studie von D. Mulfari et al. (2014) [39] wurde das Experiment nochmal wiederholt. Dieses mal wurde als Kommunikationsprotokoll RDP anstatt VNC verwendet. RDP unterstützt nativ die Umleitung von Audio. Beim Performance-Test waren die Ergebnisse mit einer durchschnittlichen Audioverzögerung von 495 ms deutlich besser als beim vorherigen Experiment. Bezüglich einer Druckeroption für Guacamole konnte keine Literatur gefunden werden. In einigen Arbeiten wird erwähnt, dass drucken für Guacamole mit RDP generell möglich ist.

## 2.3 Lösungen für eine Druckoption

Um eine Druckoption in das System zu integrieren gibt es verschiedene Möglichkeiten. Das Endziel dabei ist, ein Dokument aus dem Remote-Desktop mit einem lokalen Drucker auszudrucken. Dabei kann das ganze entweder so umgesetzt werden, dass die Druckaufträge direkt an den Drucker geleitet werden, oder dass der Druckauftrag zu einem PDF umgewan-

delt wird, der an den Nutzer geschickt wird und dann ausgedruckt werden kann. Dabei ist zu beachten, dass das ganze so einfach wie möglich für den Benutzer sein soll.

Eine Möglichkeit um das direkte Drucken ohne den PDF Umweg umzusetzen ist das Internet Printing Protokoll [57]. Dieser kann benutzt werden um mit Druckern über das Netzwerk zu kommunizieren. Jeder Drucker, der in das Netzwerk freigegeben wird, bekommt eine eigene Adresse, über die er Druckaufträge erhalten und erledigen kann. In den Einstellungen von den Remote-Desktop kann ein solcher Drucker hinzugefügt werden, indem seine Adresse eingegeben wird. Das Problem bei diesem Ansatz ist, dass der Drucker in das selbe Netzwerk wie der Remote Desktop sein. Das könnte zwar in einer Büroumgebung umgesetzt werden, allerdings ist das Ganze etwas aufwendiger, wenn der Drucker sich beispielsweise bei den Benutzer zuhause befindet. Dafür müsste sich der Benutzer per VPN mit dem Netzwerk seines Remote-Desktops verbinden und den Drucker im Netz freigeben. Für Nutzer, die sich mit so etwas auskennen, wäre dies eine Option, für allgemeine Anwender könnte dies jedoch etwas aufwändiger sein. Außerdem würde dieser Ansatz das Kriterium der Vermeidung zusätzlicher Konfigurationen verletzen.

Eine Möglichkeit, um das genannte Problem zu umgehen, ist Cloud-Printing. Beim Cloud Printing wird der Drucker in einer Cloud registriert. Dort sind in der Regel auch die Druckertreiber installiert. Der Auftrag wird somit nicht direkt an den Drucker, sondern an der Cloud geschickt, die mit dem Drucker verbunden ist. Der Vorteil dabei ist, dass die Cloud von überall erreichbar ist. Das bedeutet, dass der Remote-Desktop und der Drucker sich nicht im selben Netzwerk befinden müssen. Die Voraussetzung dafür ist, dass der Drucker von den Benutzer Cloud-Printing unterstützt. Beispiele für solche Dienste sind Ezeep [17], Directprint [13] oder Microsoft Universal Print [38]. Auch viele Druckerhersteller stellen ihre eigene Cloud Dienste zur Verfügung. Das Problem bei dieser Lösung ist, dass die Druckoption im Prinzip an einen Drittanbieter überlassen wird, was in der Regel nicht kostenlos ist. Prinzipiell können die Benutzer diese Option verwenden, wenn sie bereits einen Cloud-Printing Dienst benutzen. Dafür müsste nur der benötigte Client auf den Remote-Desktop installiert werden. Für dieses Projekt wird jedoch eine Lösung benötigt, die sofort funktionieren soll ohne zusätzlichen Aufwand für den Benutzer.

Bezüglich den zweiten Ansatz, den Druckauftrag herunterzuladen und danach zu ausdrucken, existiert die Möglichkeit die integrierte Druckoption für RDP zu benutzen. RDP unterstützt die Weiterleitung von Drucker nativ. Beim Desktopbasierten Remote-Viewer kann der Druckauftrag direkt an den lokalen Drucker weitergeleitet werden. Beim dem webbasierten Viewer von Guacamole würde das allerdings nicht funktionieren, da der Viewer nicht direkt über RDP kommuniziert, sondern über das Guacamole Protokoll. Guacamole löst das Problem folgendermaßen: Die Druckaufträge von den Remote-Desktop werden an einen sogenannten Ghostscript geleitet, der von Guacd gesteuert wird. Der Ghostscript erzeugt aus dem Auftrag einen PDF, der im Anschluss über das Guacamole Protokoll an den Nutzer geschickt wird. Der Nutzer kann die PDF Datei herunterladen und wie gewohnt bei sich lokal ausdrucken. Das Hauptproblem dieser Möglichkeit ist, dass sie nur für RDP-Verbindungen verfügbar ist. VNC bietet keine Druckerweiterleitung an. Um eine solche Möglichkeit auch für VNC-Verbindungen anzubieten, gibt es noch die Option eine eigenen Druckerserver aufzusetzen. Dieser soll Druckaufträge von den Remote-Desktops erhalten, und PDFs daraus

erzeugen, die an den Nutzern weitergeleitet werden. Das ganze würde nicht über Guacamole laufen, sondern über eine zusätzliche Verbindung. Eine Herausforderung dabei besteht in der Zuordnung der Dateien, also darin, welche Datei an welchen Web-Client gesendet werden muss. Ein Vorteil dieser Lösung ist, dass sie unabhängig vom Verbindungstyp ist.

## 2.4 Lösungen für Audioübertragung

Als nächstes wird nach Lösungen für die Audioübertragung untersucht. Dabei ist zu berücksichtigen, dass die Übertragung in beide Richtungen erfolgen muss. Das bedeutet, dass die Audioausgabe des Remote-Desktops im Browser wiedergegeben werden soll und gleichzeitig soll es möglich sein Mikrofonaufnahmen vom Browser an das Remote-Mikrofon zu senden.

Eine oft verwendete Lösung dafür ist die Audioübertragungsfunktion von RDP. Ähnlich wie beim Drucken unterstützt das Protokoll auch die Audioumleitung. Das gilt sowohl für die Ausgabe als auch für die Eingabe. Das Guacamole Protokoll unterstützt ebenfalls den Transfer von Audiodaten in beide Richtungen. Der Vorteil bei dieser Lösung ist, dass dafür keine extra Verbindung aufgebaut werden muss und keine weitere Protokolle benötigt werden. Außerdem bleibt das Audio somit synchron zum Bild. Der Nachteil dabei ist, dass diese Funktionalität nur für RDP-Verbindungen verfügbar ist. Da VNC keine Audioumleitung unterstützt, wird dafür eine andere Lösung benötigt. Hierfür bietet Guacamole die Möglichkeit die Audioausgabe über Pulseaudio zu streamen. Der Guacamole Server kann sich mit dem Audio-Server von den Remote-Desktop verbinden und die Audiodaten über das Guacamole Protokoll an den Web-Client weiterzuleiten. Das funktioniert aber nur für die Audioausgabe. Für die Benutzung von Mikrofon bietet Guacamole keine Lösung an.

Neben VNC und RDP existiert noch das Simple Protocol for Independent Computing Environments (SPICE) Protokoll[53]. Es wird hauptsächlich dafür benutzt um virtuelle Maschinen fernzusteuern. Neben Maus und Tastatur unterstützt das Protokoll auch Audioübertragung. Ähnlich wie anderen Fernsteuerungsprotokolle besteht SPICE ebenfalls aus den Komponenten Server, Client und das Protokoll selbst. SPICE-Verbindungen werden von Guacamole nicht unterstützt, wobei es in zukünftige Versionen integriert werden könnte [33]. Um nur das Audio zu übertragen, müsste hier eine zweite unabhängige Verbindung von der VM zu dem Web-Client aufgebaut werden.

Eine andere Methode für die Übertragung von Audio ist Web Real-Time Communication (WebRTC). Dabei handelt es sich um eine relativ neue Technologie von Google für Echtzeit-Datenübertragung. WebRTC wird von den meisten modernen Browsern unterstützt. Es besteht aus drei Hauptkomponenten: Die MediaStream API ermöglicht den Browsern den Zugriff auf das Mikrofon und die Kamera und implementiert die entsprechenden Funktionen für Audio- und Videoaufnahme. RTCPeerConnection ermöglicht eine direkte Echtzeitkommunikation für Audio und Video zwischen zwei Browsern. Prinzipiell ist auch eine direkte Verbindung zwischen einem Browser und einem Desktop-Verbindung möglich. RTCDataChannel repräsentiert eine bidirektionale Echtzeitkommunikation zwischen zwei Geräten für den Transfer von beliebigen Daten wie Spielen oder Chats. Eine direkte Peer-to-Peer-Verbindung

aufzubauen ist allerdings etwas aufwendig, weil die meisten Geräte hinter sogenannte NATs (Network Address Translation) liegen. Das ist ein Verfahren, das private IP-Adressen in eine Öffentliche umwandelt. Während Datenpakete durch ein Gerät geleitet werden, ändert NAT die Adressinformationen, damit sie korrekt an ihren Ziel im Internet weitergeleitet werden können. Das erschwert das Prozess, dass sich die zwei Peers finden. Um das Problem zu lösen, nutzt WebRTC das ICE (Interactive Connectivity Establishment) Konzept um die Peers zu verbinden. Dabei wird einen STUN (Session Traversal Utilities for NAT) Server verwendet um die öffentliche IP und Port des Geräts zu ermitteln. Falls das nicht funktioniert wird der Datenverkehr über einen TURN-Server geleitet, was eventuell zur Latenz führen kann. [54]

Eine weitere Option ist den PulseAudio Server sowohl für die Audioeingabe als auch Ausgabe zu benutzen. Wie bereits beschrieben, kann Guacamole die Audioausgabe aus dem PulseAudio Server erhalten und über die Guacamole Verbindung an den Web-Client weiterleiten. Das selbe könnte im Prinzip auch in die andere Richtung gemacht werden. Das müsste aber über eine zweite Verbindung gemacht werden, da es dafür keine Unterstützung von Guacamole gibt. So könnte z.B das Audio im Browser mithilfe von eingebauten APIs aufgenommen werden, über eine Websocket-Verbindung an den Proxy gesendet werden und von dort an den PulseAudio Server geleitet werden. Bei dieser Lösungen könnte es eventuell dazu kommen, dass Verzögerungen entstehen, oder dass Bild und Audio nicht synchron sind.

## 2.5 Verwendete Technologien

In diesem Unterkapitel werden die Technologien beschrieben, die für die Realisierung des Projekts verwendet wurden. Ihre Funktionsweise und ihre spezifische Rolle in dieser Arbeit werden dabei erläutert. Sie sind in drei Abschnitten unterteilt. Der erste Abschnitt beschäftigt sich mit den Services, die von Open-Source-Anbietern zur Verfügung gestellt wurden. Der zweite Abschnitt umfasst die verwendeten Protokollen, die zur Kommunikation zwischen den Komponenten verwendet werden. Im dritten Abschnitt werden die Webtechnologien beschrieben, mit denen die Webanwendung entwickelt wurde.

### 2.5.1 Services

Im Folgenden werden die Open-Source-Dienste und Plattformen vorgestellt, die für die Realisierung des Projekts benutzt wurden. Dazu gehören Proxmox VE und Docker für die Erstellung und Verwaltung von VMs und Containern, das Apache Guacamole Projekt für Darstellung des Desktops im Browser, PulseAudio für die Audioverwaltung unter Linux sowie CUPS für die Erstellung eines Druckservers.

### 2.5.1.1 Proxmox

Proxmox Virtual Environment (PVE)[9, 46] ist eine Open-Source Virtualisierungsplattform für Servern. Es basiert auf der Debian GNU/Linux Distribution und dient zur Verwaltung von virtuelle Maschinen, Containern, Speicher, virtuelle Netzwerke und Hochverfügbarkeits-Cluster. Die Verwaltung erfolgt entweder über die webbasierte Benutzerschnittstelle oder über die Kommandozeile. PVE ist ein Hypervisor vom Typ 2, da die Virtualisierungsschicht auf das Betriebssystem liegt. Zum Vergleich, ein Typ 1 Hypervisor läuft direkt auf die Hardware ohne ein unterliegendes Betriebssystem (Siehe Abb. 2.1). Als unterliegendes Betriebssystem für Proxmox wird Debian verwendet, da es als stabil, sicher und zuverlässig gilt. Außerdem bietet Debian regelmäßige Sicherheitsupdates und Langzeitunterstützung.

PVE bietet zwei Virtualisierungsmöglichkeiten an. Die erste sind Virtuelle Maschinen (VM)s. Um sie auszuführen, wird das Kernel-based Virtual Machine (KVM) Modul verwendet. Es ist Teil des Linux Kernels und nutzt die Hardware-Virtuallisierungstechniken des Prozessors. Damit wird die Ausführung von mehrere VMs ermöglicht. KVM alleine emuliert keine Hardware. Dazu ist der Quick Emulator (QEMU)[45] zuständig. Das ist ein Open-Source Hypervisor, der die Hardware von einem echten Computer emuliert. Somit können Betriebssysteme laufen, die für eine andere Architektur konzipiert sind. Abbildung 2.2 zeigt wie KVM und QEMU zusammenhängen. Auf der Hardware-Ebene läuft der Linux Kernel mit dem integrierten KVM-Treiber. Darauf werden die normalen Prozesse des Host-Systems und die VMs ausgeführt. QEMU stellt die virtuellen Geräte und die Schnittstellen für die VMs bereit. Die zweite Virtuallisierungsmethode sind Linux Container (LXC). Im Gegensatz zu Vms, die ein eigenen Kernel emulierte Hardware benötigen, teilen sich LXC-Container den Kernel mit dem Host-System. Dadurch wird der Ressourcenverbrauch stark reduziert, da weniger Speicher und Rechenleistung benötigt wird. So können mehrere Container parallel ausgeführt werden, ohne die Leistungsfähigkeit des Host-Systems stark zu beeinträchtigen.

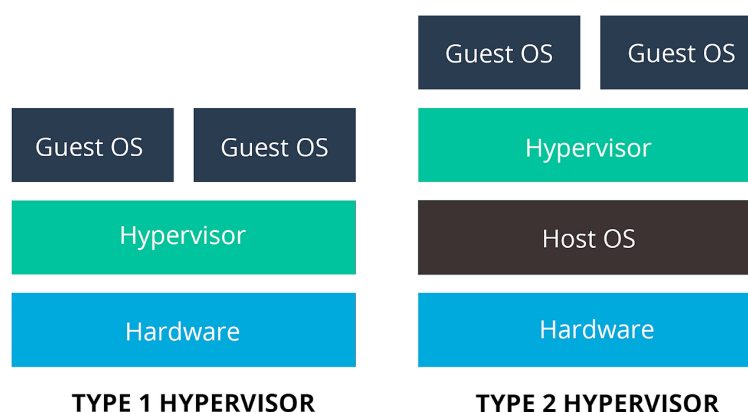


Abbildung 2.1: Hypervisor Typ 1: direkt auf der Hardware, Typ 2: benötigt Host-OS

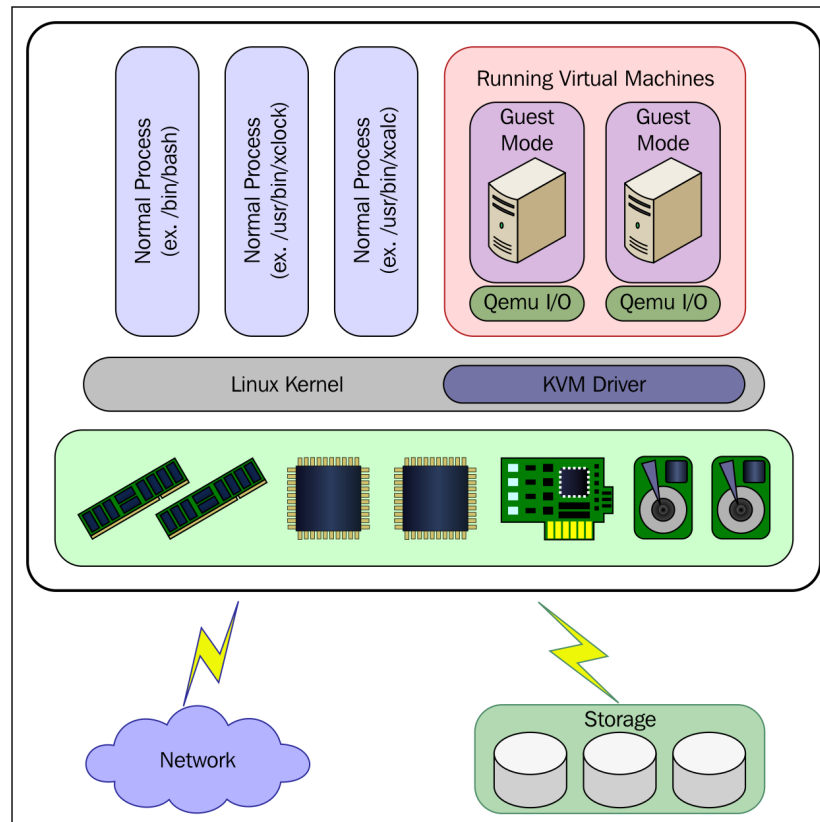


Abbildung 2.2: Architekturdiagramm zur Ausführung von VMs unter Linux

Für diese Arbeit wird PVE als Host-System zur Verwaltung der benötigten VMs eingesetzt. Durch die benutzerfreundliche Weboberfläche können VMs einfach erzeugt, überwacht und konfiguriert werden. Durch den Einsatz von Proxmox kommt die Architektur des Projekts näher an der von dem DESIGN-Projekt, das die gleiche Plattform für die Bereitstellung und Verwaltung von VMs nutzt.

### 2.5.1.2 Apache Guacamole

Das Apache Guacamole Projekt [26] ist ein Open-Source Projekt, der die Fernsteuerung von Computer-Desktops im Browser ermöglicht. Es unterstützt verschiedene Protokolle für Fernsteuerung wie RDP, VNC oder SSH. Guacamole kann von jedem internetfähigen Gerät mit einem aktuellen Browser genutzt werden, ohne dass zusätzliche Installationen erforderlich sind. Das Projekt besteht aus zwei Teilen, den Guacamole Server (Guacd) und einer Webanwendung. [26]

Guacd ist das Herzstück des Projekts und dient als Vermittler zwischen den Client und den Remote-Computer. Er ist in C geschrieben und läuft als Hintergrundprozess, der auf TCP-Verbindungen wartet. Seine Hauptaufgabe ist Protokolle wie VNC oder RDP in einen Textbasierten Protokoll (siehe Abschnitt 2.5.2.1) zu übersetzen und an den Client weiterzuleiten. Außerdem verwaltet er die verschiedenen Sitzungen. Guacd selbst, versteht keine

spezifische Protokolle, sondern lädt für jedes Protokoll einen sogenannten "Client Plugin", der die Kommunikation zwischen dem Remote-Desktop und Client übernimmt. Die Webanwendung besteht aus einem Web-Client (Frontend), einem Server (Backend) und optional eine Datenbank zur Authentifizierung. Der Server ist in Java geschrieben und läuft als Apache Tomcat Servlet-Container. Er dient als Vermittler zwischen Guacd und dem Web-Client. Seine Aufgaben sind die Verwaltung von Sitzungen, Nutzer Authentifizierung und die Weiterleitung von den Kommunikationsdaten. Der Web-Client ist in JavaScript, HTML5 und CSS geschrieben. Seine Aufgabe ist den Remote-Desktop im Browser darzustellen und die Interaktion mit dem Nutzer zu ermöglichen. Der Web-Client kommuniziert über HTTP mit dem Server, während der Server über TCP mit Guacd kommuniziert. Die Architektur des Projekts (Abbildung 2.3) sieht wie folgt aus: Die Remote-Geräte kommunizieren mit guacd über beliebige Protokolle für Fernsteuerung. Guacd leitet die erhaltenen Daten an der Client-Anwendung, die als Servlet-Container ausgeführt wird, über das Guacamole-Protokoll. [24]

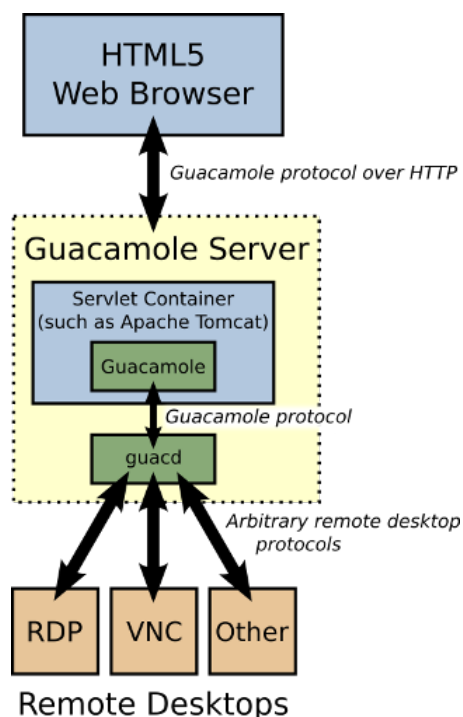


Abbildung 2.3: Architektur des Apache Guacamole Projekts

Beide Teile des Projekts sind unabhängig voneinander. Die Komponenten sowie die einzelnen Bibliotheken stehen frei zur Verfügung. Für diese Arbeit wird nur der Guacamole-Server (guacd) verwendet, um die Daten der Fernsteuerungsprotokolle im Browser darzustellen. Der Client wird nicht benötigt, da der Remote-Desktop-Viewer in eine eigenen Anwendung integriert werden soll.

### 2.5.1.3 CUPS

Das Common Unix Printing System (CUPS) [61] ist das am meisten verbreitetes Drucksystem für Unix-basierte Systeme. Das System ermöglicht die Nutzung, Verwaltung und Freigabe von Druckern im Netzwerk. Es hat ältere Systeme wie LPD oder lprng größtenteils verdrängt. CUPS ist ein wichtiger Bestandteil des Linux-Betriebssystems und wird bei Ubuntu-basierte Systeme automatisch mitinstalliert.

CUPS wandelt die zu druckenden Daten durch bestimmte Filter zu PDFs um, die dann an den Druckinstanzen weitergeleitet werden. Die Instanz, die den Drucker steuert, muss die PDF-Druckdaten entweder über Treiber in gerätespezifische Druckdaten umwandeln oder, falls das Gerät PDF direkt unterstützt, die Daten direkt an den Drucker senden. Zusätzlich werden .ppd Dateien benötigt, die die Features des Druckers beschreiben. Falls es für den spezifischen Drucker keine .ppd-Datei gibt, kann es mit einer generischen .ppd-Datei von CUPS versucht werden. CUPS kann entweder über die Kommandozeile oder über die Weboberfläche benutzt werden. Die Weboberfläche ist auf Port 631 zu erreichen. Von dort können Drucker, Aufträge, Warteschlangen und weitere Einstellungen verwaltet werden. Es ist möglich Drucker das Netzwerk freizugeben, das heißt andere Rechner in den selben Netzwerk können freigegebenen Drucker benutzen. Um festzulegen welcher Nutzer welche Rechte auf dem Server besitzt, verwendet CUPS Access Control Lists (ACL). Jeder Bereich des Servers wird durch virtuelle Verzeichnisse dargestellt und kann unterschiedliche Berechtigungen haben. Die ACL sind in der Datei `/etc/cups/cupsd.conf` definiert.

Der CUPS-Server wird für diese Arbeit benutzt, um das Drucken für VNC-Verbindungen zu ermöglichen. Dabei wird einen Server eingerichtet, der für jeden Nutzer einen PDF-Drucker verwalten wird. Die VMs werden deren Aufträge an den CUPS-Server über das IPP-Protokoll (Abschnitt 2.5.2.4) senden, wo sie mithilfe von den PDF-Druckern zu PDF-Dateien umgewandelt werden.

### 2.5.1.4 PulseAudio

PulseAudio [28] ist ein Sound-Server für Linux Systeme, der als Proxy und Router zwischen Hardware und Anwendungen dient. Er bietet eine Abstraktionsschicht für den gesamten Audio-Bereich des Desktops, über die Audioanwendungen, Geräte, Filter und Ein-/Ausgaben zentral verwaltet werden können. Eine umfangreiche API erlaubt eine detaillierte Steuerung und Inspektion, sodass die Konfiguration oft über die grafische Oberflächen statt über manuelle Dateien erfolgen kann. PulseAudio ist so konzipiert, dass es sich von alleine einrichtet und alle lokale und Netzwerk-gebundene Geräte automatisch erkannt und eingerichtet werden. Für den Nutzer wird eine hohe Flexibilität angeboten. Es ist beispielsweise möglich beliebige Streams an beliebige Geräte zu leiten, Lautstärken anzupassen oder Audio-Verarbeitungsketten zu erstellen. PulseAudio bietet ein Framework für Erweiterungen. Viele Funktionalitäten sind als Build-in Module vorhanden. Es existieren auch Drittanbieter-Module, für die es allerdings keine Garantie gibt.

PulseAudio bietet vielseitige Funktionen für die Audioverwaltung, darunter Netzwerkunterstützung, automatische Gerätekonfiguration und Soundverarbeitung-Tools wie Mixen und



Echounterdrückung. Es verfügt über einen Cache, verwaltet alle Streams und bietet ihnen Funktionen wie Taktung, Pufferung und Zurückspulen, und sorgt für präzise Taktung und niedrige Latenz. Dazu ist PulseAudio energiesparend, setzt sich automatisch auf, integriert sich leicht in Desktopumgebungen und ermöglicht durch Kompatibilitätsschichten die Nutzung bestehender Anwendungen ohne Anpassungen.

Beim PulseAudio gibt es zwei Arten von Geräten: "sources" und "sinks" (Abbildung 2.4). Beim Sources handelt es sich um Eingabegeräte wie Mikrofon. Diese laufen in eigene Threads und erzeugen Audiodaten, die an verbundene Ausgänge (source outputs) gesendet werden. Für jedes erkanntes Eingabegerät wird automatisch einen Source erzeugt. Die Source Outputs repräsentieren Aufnahme-Streams die an einen Sink angeschlossen sind und werden von Anwendungen benutzt. Sinks repräsentieren Ausgabegeräte wie Kopfhörer oder Lautsprecher. Sie laufen in eigene Threads, konsumieren Daten von den Sink Inputs und mischen diese. Für jedes erkanntes Ausgabegerät wird automatisch einen Sink erzeugt. Sink Inputs repräsentieren Wiedergabe-Streams die Audiodaten produzieren und an den Sinks leiten. Sie werden von den Anwendungen benutzt.

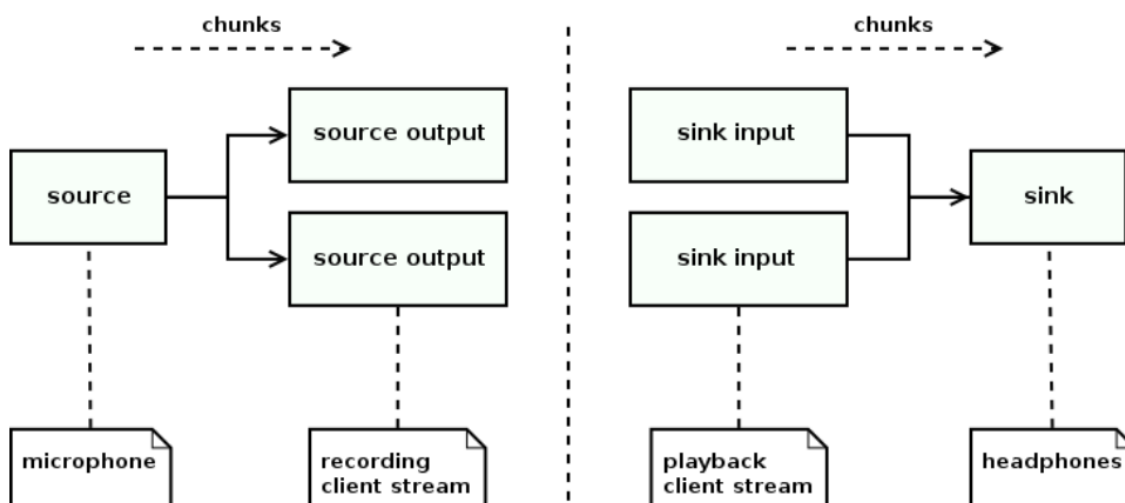


Abbildung 2.4: Überblick von Sources und Sinks unter PulseAudio

PulseAudio wird für diese Arbeit benutzt um das Audio bei Linux-VNC-Verbindungen an den Webclient zu übertragen. Durch die Unterstützung von TCP Verbindungen, können Audiodaten über mit PulseAudio über das Netzwerk empfangen und gesendet werden. Mithilfe von den Sources, Sinks und Tools wie pacat können Audio-Streams beliebig umgeleitet werden.

### 2.5.1.5 Docker

Docker [14] ist eine Open-Source Technologie zur Erstellung von Containern. Container stellen isolierte Umgebungen dar, wo alles Nötigste enthalten ist um Anwendungen zu laufen. Sie können auf alle Betriebssysteme laufen solange Docker drauf installiert ist. Docker basiert auf bekannte Technologien aus dem Bereich der Betriebssysteme wie LXC-Kontainern, Virtualisierung des Betriebssystems und ein Hash-basiertes oder git-ähnliches Versionierungs- und Differenzierungssystem.

Einer der größten Vorteile von Docker ist, dass es Nutzern einfach macht Projekte zu reproduzieren. Oft ist es für Forscher kompliziert an Software-Projekte weiterzuforschen da es schwer ist das ursprüngliche Projekt zu reproduzieren. In einem Paper beschreibt C. Boettiger [5] die Gründe warum das schwer ist und wie Docker das Problem löst. Vier Herausforderungen werden genannt: Die Erste Herausforderung ist das sogenannte "Dependency Hell". Das beschreibt das Problem, dass in die neue Umgebung oft wichtige Abhängigkeiten fehlen, die die Anwendung benötigt um zu laufen. Die zweite ist die ungenaue Dokumentation. Es kann vorkommen, dass bei Anleitungen etwas vergessen wird. Auch fehlenden Kleinigkeiten können den Installationsprozess zum Scheitern bringen. Die dritte Herausforderung ist das "Code Rot" Problem. Abhängigkeiten sind nicht statisch. Sie erhalten Updates und ändern sich ständig. Das kann dazu führen, dass der Code mit neuere Versionen nicht mehr richtig läuft. Die vierte Herausforderung besteht im fehlenden Wissen über spezifische Technologien. Oft ist es erforderlich, sich mit bestimmten Technologien auszukennen um Projekte zu reproduzieren, über die Forscher nicht ausreichend Wissen oder Zeit zum Erlernen haben.

Docker löst viele dieser Herausforderungen und vereinfacht das Problem bestehender Software wiederzuverwenden. Seine wichtigsten Konzepte sind Docker-Images, Dockerfiles und Image Versionen. Die Docker-Images sind ähnlich wie die VM-Images. Aus denen werden die Containern erzeugt. Dort ist alles installiert und konfiguriert, was für die Anwendung benötigt wird. Dadurch wird das Problem gelöst, dass Nutzer alle Abhängigkeiten installieren müssen um die Anwendung zu starten ("Dependency Hell"). Der Hauptunterschied zwischen Docker- und VM-Images ist, dass Docker sich den Linux Kernel mit dem Host-System teilt, während bei VMs das ganze Betriebssystem installiert werden muss. Falls das Hostsystem Windows ist, muss das Windows-Subsystem für Linux (WSL) [15] installiert werden. Docker ist sehr effizient und es können problemlos hunderte Containern gleichzeitig laufen. Dockerfiles sind Skripte, aus denen Docker-Images erzeugt werden. Es sind einfache Textdateien, die z.B über git verwaltet werden können. In den Dockerfile sind alle Abhängigkeiten, Umgebungsvariablen und andere Konfigurationen definiert, die benötigt werden um die Anwendung zu starten. Durch diesen Ansatz werden keine Abhängigkeiten vergessen. Anders als bei Makefile, werden bei Dockerfile alle Abhängigkeiten bis zu der Betriebssystem-Ebene definiert. Somit unterscheiden sich Images die auf verschiedenen Maschinen erzeugt werden kaum voneinander. Es können auch Überprüfungen und Tests hinzugefügt werden, um sicherzustellen, dass alles richtig funktioniert. Dockerfiles können einfach von anderen Entwickler erweitert werden. Auch wenn Docker das Problem mit der Umgebung und Abhängigkeiten gut löst, bleibt trotzdem das sogenannte "Code Rot" Problem. Das bedeutet, dass der Code über die Zeit kaputt gehen kann durch Updates von den Abhängigkeiten, Entfernung von veralteten Bibliotheken und andere Änderungen. Um das zu Umgehen bietet Docker die Möglichkeit

Images als "tarball" Dateien zu speichern, die auf Dockerhub veröffentlicht werden können. So können Images direkt geladen werden und daraus Container erzeugt werden, ohne dass das Image zuerst aus dem Dockerfile gebaut wird.

Docker wird in dieser Arbeit verwendet, um die einzelnen Komponenten als Containern auszuführen. So würde jede Komponenten eine eigene isolierte Umgebung bekommen, in der alle erforderlichen Abhängigkeiten installiert sind. Außerdem kann somit die Anwendung sehr einfach auf jedes Gerät, wo Docker installiert ist, ausgeführt werden.

## 2.5.2 Protokolle

In diesen Abschnitt werden die Protokolle beschrieben, mit denen die verschiedenen Komponenten der Anwendung miteinander kommunizieren. Dazu zählen RDP und VNC zur Fernsteuerung und Übertragung von Desktops, Guacamole für die Kommunikation zwischen guacd und den Browser, IPP für Übertragung von Druckaufträge über das Netzwerk und Websocket für die bidirektionale Kommunikation zwischen den Backend und der Webbrowser.

### 2.5.2.1 Guacamole Protokoll

Das Guacamole Protokoll [25] ist ein textbasiertes Protokoll, das zur Kommunikation zwischen guacd und den Client dient. Es besteht aus Anweisungen, die mit Semikolon getrennt sind. Jede Anweisung besteht wiederum aus Elementen, die mit Kommas getrennt sind. Das erste Element ist ein Instruktions-Opcode, während die nachfolgenden Elemente Argumente darstellen. Ein Element besteht aus einer Länge und einen Wert, die mit Punkt getrennt sind. Der Wert ist eine UTF-8 codierte Unicode Zeichenkette und die Länge ist die Anzahl der Zeichen. Durch diese Struktur kann der Datenstrom von Anweisungen leicht geparkt werden, ohne dass über jedes einzelne Zeichen iteriert werden muss. Listing 1 zeigt auf Zeile 2 ein Beispiel für eine einfache Anweisung. Der OP-Code in diesen Beispiel ist `size` (Größe), gefolgt von "0", der Index der Standard-Ebene, "1024", die Bildschirmbreite in Pixeln, und "768", die Bildschirmhöhe in Pixeln. Zu jedem Wert steht auch die jeweilige Länge.

Jede Verbindung beginnt mit einer Handshake Phase. Diese Phase ist folgendermaßen aufgebaut: Als erstes wird das Protokoll für Fernzugriff ausgewählt. Um z.B. VNC zu wählen wird eine `select`-Anweisung an guacd geschickt (Listing 1, Zeile 5). Guacd lädt den entsprechenden Client-Plugin für das Protokoll und schick an den Client eine `args`-Anweisung (Zeile 8), die die Protokollversion und die akzeptierenden Parameter enthält. Falls die Protokollversionen von guacd und den Client nicht kompatibel sind, wird die Verbindung abgebrochen. Als nächstes schickt der Client eine Liste mit den unterstützten Formaten für Audio, Video, Bilder und andere Parameter. Die letzte Anweisung ist `connect`, die die Verbindungsparameter wie z.B. IP-Adresse der Maschine, Nutzernamen und Passwort enthält. Zeilen 10-16 zeigen wie solche Anweisungen aussehen können. Guacd versucht die Verbindung zu den Remote-Desktop mit den gegebenen Parameter aufzubauen. Wenn das klappt, schickt er eine `ready`-Anweisung mit der Verbindungs-ID an den Client, sodass er die Sitzung bei-

treten kann (Zeilen 19-20). Nachdem der Handshake abgeschlossen ist, kann der Benutzer seinen Desktop im Browser benutzen.

```
1 # size-Anweisung
2 4.size,1.0,4.1024,3.768;
3
4 # select-Anweisung
5 6.select,3.vnc;
6
7 # args-Anweisung
8 4.args,13.VERSION_1_1_0,8.hostname,4.port,8.password,13.swap-red-blue,
   ↪ 9.read-only;
9
10 # Unterstützte Formate
11 4.size,4.1024,3.768,2.96;
12 5.audio,9.audio/ogg;
13 5.video;
14 5.image,9.image/png,10.image/jpeg;
15 8.timezone,16.America/New_York;
16 7.connect,13.VERSION_1_1_0,9.localhost,4.5900,0.,0.,0.;
17
18 # Verbindung ist fertig
19 5.ready,37.$260d01da-779b-4ee9-afc1-c16bae885cc7;
20 6.select,37.$260d01da-779b-4ee9-afc1-c16bae885cc7;
```

Listing 1: Beispiele für Guacamole-Protokoll-Anweisungen

### 2.5.2.2 RDP Protokoll

Das Remote Desktop Protokoll (RDP) [37] wurde von Microsoft entwickelt und dient zur Fernsteuerung von Computern über das Netz. RDP basiert auf der T.120-Protokollfamilie und unterstützt bis zu 64.000 separate Kanäle für die Datenübertragung. Das Protokoll basiert auf dem Client-Server Prinzip. Der Server verwendet einen eigenen Videotreiber, um die Anzeigerausgabe zu rendern und sendet diese als Netzwerkpakete an den Client. Der Client empfängt und interpretiert die Pakete in Graphics Device Interface (GDI)-API Aufrufe und leitet Maus- und Tastaturereignisse zurück an den Server, wo sie von speziellen Treibern verarbeitet werden. Umgebungsvariablen wie Farbtiefe und Hintergrundbild werden durch die RDP-TCP-Verbindungseinstellungen bestimmt. RDP verfügt über folgende Funktionalitäten: Es nutzt RC4-Verschlüsselung mit einem 56- oder 128-Bit-Schlüssel. Zur Bandbreitenreduzierung werden Datenkomprimierung, persistenter Bitmap-Cache und Zwischenspeicherung von Glyphen und Fragmenten verwendet. Nutzer können Sitzungen trennen und später wieder aufnehmen, ohne sich abzumelden. Das Kopieren und Einfügen von Texten und Graphiken zwischen den lokalen und fernen Computer ist möglich. Anwendungen von

den Fern-Computer können Druckaufträge an den lokalen Drucker weiterleiten. Durch virtuelle Kanäle können bestehende und neue Anwendungen erweitert werden, um Funktionen zu integrieren, die die Kommunikation zwischen dem Clientgerät und einer Anwendung in einer Remotedesktopsitzung ermöglichen. Supportmitarbeiter können Remote-Sitzungen anzeigen und steuern. RDP unterstützt ebenfalls Network Load Balancing, 24-Bit-Farben, optimierte Leistung für Low-Speed-Verbindungen, Smartcard-Authentifizierung, Tastaturhaken sowie Sound-, Laufwerk-, Port- und Netzwerkdruckerumleitung.

### 2.5.2.3 VNC Protokoll

Das Virtual Network Computing (VNC) Protokoll von T. Richardson et al. [50] ist ein leichtgewichtiges Protokoll für Fernzugriff von graphische Benutzeroberflächen. Es wird als ein "thin-client" System betrachtet, da die Systemanforderungen für den Client sehr gering sind und der Großteil der Verarbeitung auf dem Server stattfindet. Bei der Architektur handelt es sich ebenfalls wie beim RDP um eine Client-Server-Architektur. Das Protokoll arbeitet auf der Framebuffer-Ebene und funktioniert deswegen für alle Betriebssysteme. Im Gegensatz zu RDP bietet VNC nur die nötigsten Funktionen, um einen Desktop fernzusteuern, wie Maus, Tastatur und Bild. Funktionen wie Drucken, Audio oder Dateitransfer werden nicht unterstützt. Es gibt viele Implementierungen des VNC-Konzepts, wie z.B. RealVNC, UltraVNC, TightVNC, TigerVNC und x11vnc.

### 2.5.2.4 Internet Printing Protokoll

Das Internet Printing Protocol (IPP) [57] ist ein Protokoll, das auf der Anwendungsebene arbeitet und die Kommunikation mit Druckern über das Netzwerk ermöglicht. Es basiert auf einem vereinfachten Druckmodell, das nur die wesentlichen Objekte und Schnittstellen für das Drucken im Internet abstrahiert, obwohl die zugrunde liegenden Systeme komplex sein können. Der Fokus liegt hauptsächlich auf der Endnutzerfunktionalität, während einige optionale Verwaltungsoperationen ebenfalls unterstützt werden.

Das IPP-Model (Abbildung 2.5) kapselt die wichtigsten Komponenten in den Objekten Printer, Job, Document und Subscription. Ein IPP Printer Objekt umfasst Funktionen, die direkt mit dem physischen Ausgabegerät sowie mit dem Drucker-Server verbunden sind sowie wie Spooling, Scheduling, und Geräteverwaltung. Sie können optional in Verzeichnissen eingetragen werden wo sie von Nutzern nach Kriterien gefiltert werden können. Dynamische Informationen wie Status oder Anzahl von Aufträge sind direkt mit dem Druckerobjekt verknüpft und nicht mit den Verzeichnis-Einträgen. IPP Clients ermöglichen es Benutzern, Drucker abzurufen und Druckaufträge zu versenden. Der IPP-Server ist Teil des Printer-Objekts und implementiert die Server-Seite des Protokolls. Wenn ein Nutzer einen Druckauftrag (Job) an einen Drucker schickt, wird ein Job-Objekt erzeugt. Der Nutzer kann mit diesen Objekt interagieren um beispielsweise den Status abrufen oder den Auftrag abubrechen. Administratoren haben die volle Kontrolle über alle Jobs und Drucker. Durch den Benachrichtigungsservice können Nutzer automatisch über Drucker- und Job Ereignisse informiert werden.

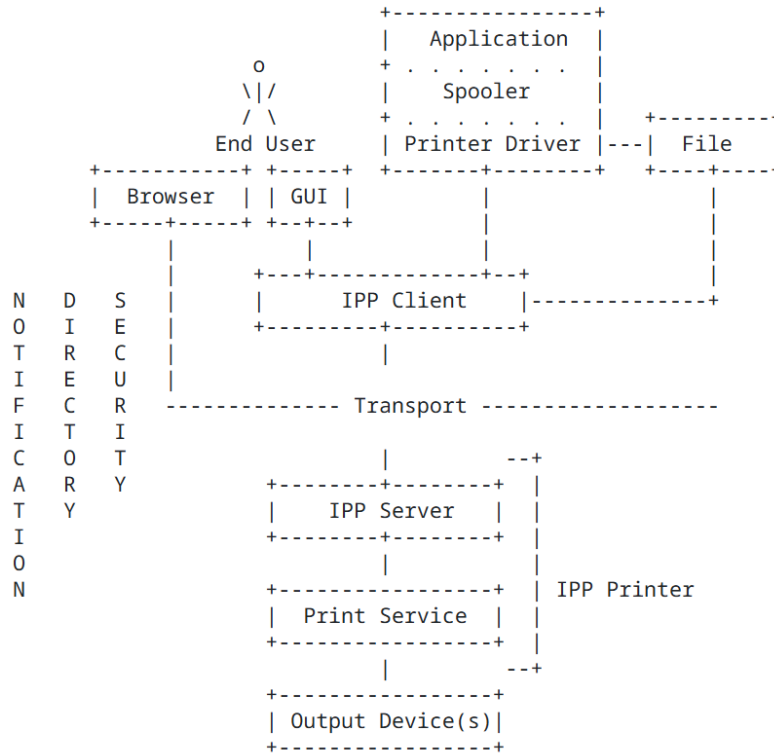


Abbildung 2.5: Design des IPP-Models

Die Kommunikation zwischen den Clients und Servern verläuft über IPP-Operationen, die über HTTP-Anfragen transportiert werden. Eine IPP-Operation besteht aus einer Anfrage und einer Antwort. Wenn ein Client eine Anfrage an einen Drucker oder einen Job senden möchte, benötigt er dafür die Drucker-URI bzw. Job-URI und falls erforderlich auch die ID des Zielobjekts. Jede Anfrage enthält Operationsattribute, Objektattribute und gegebenenfalls Dokumentdaten. Die Antworten enthält Informationen wie, ob die Operation erfolgreich war, den Status und weitere Parameter. Wenn der Client z.B ein Dokument ausdrucken möchte, sendet er eine Anfrage an den Drucker mit der Operation Print-Job. Dabei werden Parameter wie Job-Name, Dokumentname, Dokumentformat, das Dokument selbst und Komprimierungsalgorithmus übermittelt. Als Antwort erhält der Client Informationen wie den Status, die Job-ID, die Job-URI und Weitere.

Für diese Arbeit wird IPP benutzt um Druckaufträge von den Linux-VMs an den CUPS-Server zu senden, da VNC keine Druckaufträge übertragen kann. Windows kann ebenfalls problemlos Druckaufträge über IPP an den Server senden, allerdings wird bei Windows hauptsächlich das RDP-Protokoll benutzt wo das Ganze nicht benötigt wird, wegen der eingebauten Druckoption.

### 2.5.2.5 WebSocket

Das WebSocket Protokoll [36] stellt eine bidirektionale Kommunikationsverbindung zwischen einen Client und einen Server zur Verfügung. Der Hauptvorteil dabei ist, dass beide Seiten jederzeit Nachrichten miteinander austauschen können sobald die Verbindung aufgebaut ist. Um eine solche Kommunikation aufzubauen wurden davor andere Techniken wie Polling[51] verwendet, die auf HTTP Aufrufe basieren. Das Problem dabei ist, dass HTTP Aufrufe auf das "Anfrage-Antwort" Prinzip basieren. Das heißt der Server kann nur dann Daten an den Client schicken kann, wenn eine entsprechende Anfrage vom Client ankommt. Allerdings weiß der Client nicht ob Daten für ihn zur Verfügung stehen. Um das zu umgehen wurden beispielsweise Techniken Short-Polling und Long-Polling verwendet. Beim Short-Polling schickt der Client in bestimmte Zeitabstände Anfragen an den Server um Daten abzurufen. Wenn keine Daten für ihn vorhanden sind, kommt eine leere Antwort zurück und die Anfrage wird zu einem späteren Zeitpunkt wiederholt. Das ganze ist allerdings ineffizient, da es viele Ressourcen verbraucht und die Latenz nicht niedrig ist. Beim Long-Polling schickt der Client eine Anfrage an den Server. Der Server lässt die Verbindung offen und wartet so lange bis Daten für den Client verfügbar sind. Wenn es soweit kommt, werden die Daten als Antwort an den Client geschickt und dann wird die nächste Long-Poll Anfrage geschickt. HTTP wurde grundsätzlich nicht für solche Art von Kommunikationen entwickelt. Daraus resultieren Probleme wie beispielsweise, dass der Server mehrere TCP Verbindungen für jeden Client aufrechterhalten muss, hoher Overhead durch HTTP Header oder Latenz. Eine elegantere Lösung wäre eine einzelne TCP-Verbindung für beide Richtungen zu benutzen. Dafür wurde das WebSocket Protokoll [36] entwickelt. Es ist bidirektional und der Server kann jederzeit Daten an den Client senden, ohne eine entsprechende Anfrage.

Eine WebSocket Verbindung beginnt mit einem Handshake. Dabei schickt der Client eine sogenannte HTTP Upgrade Anfrage (Listing 2), die den Übergang von HTTP zu WebSocket signalisiert. Wenn der Server die Anfrage akzeptiert, schickt er als Antwort den Status 101 Switching Protocols zurück (Listing 3).

```
1 GET /chat HTTP/1.1
2 Host: server.example.com
3 Upgrade: websocket
4 Connection: Upgrade
5 Sec-WebSocket-Key: dGh1IHNhbXBsZSBub25jZQ==
6 Origin: http://example.com
7 Sec-WebSocket-Protocol: chat, superchat
8 Sec-WebSocket-Version: 13
```

Listing 2: Client-Anfrage zum Wechsel zu WebSocket

```
1 HTTP/1.1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
5 Sec-WebSocket-Protocol: chat
```

Listing 3: Bestätigung vom Server zum Wechsel zu WebSocket

Nachdem die Handshake Phase abgeschlossen ist, können beide Seiten Nachrichten miteinander austauschen. Abbildung 2.6 zeigt die Kommunikation graphisch. Die Nachrichten werden als Frames geschickt. Dabei gibt es textbasierte Frames, binäre Frames und Steuerungframes. Um sicherzustellen, dass die Verbindung aktiv bleibt werden Ping und Pong Nachrichten ausgetauscht. Die Verbindung bleibt so lange aktiv bis eine der beiden Seiten sie schließt. WebSockets werden oft beim Echtzeitanwendungen angewendet, wo regelmäßig Daten in beide Richtungen gesendet werden. Ein Beispiel dafür ist das Guacamole-Protokoll, dass auf einer WebSocket Verbindung basiert.

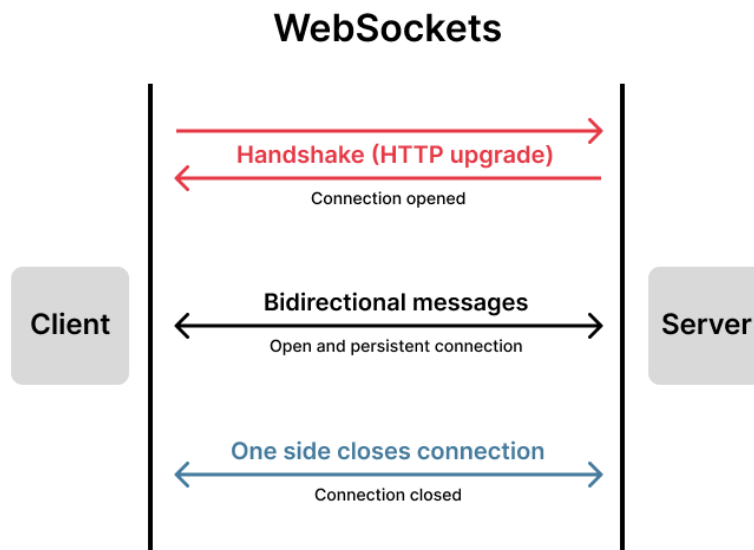


Abbildung 2.6: Verbindungsaufbau und Nachrichtenaustausch bei WebSocket Verbindungen

In dieser Arbeit werden WebSockets verwendet, um die Guacamole-Protokolldaten zwischen dem Frontend und Backend zu übertragen. Das WebSocket-Protokoll bietet zudem die Möglichkeit, zusätzliche Verbindungen zu ermöglichen und Daten in beiden Richtungen zu übertragen, die vom VNC-Protokoll nicht unterstützt werden.



## 2.5.3 Webtechnologien

Dieser Abschnitt befasst sich mit den Webtechnologien, die für die Entwicklung der Webanwendung benutzt wurden. Dazu gehören die JavaScript-Serverseitige-Umgebung Node.js sowie die Frameworks Express.js und React. Express wird für das Backend eingesetzt und React für das Frontend. Abschließend wird erklärt wie die Verbindungen zwischen Backend und Frontend mithilfe von TLS verschlüsselt werden können.

### 2.5.3.1 Node.js und Express

Node.js [12] ist eine Open-Source, plattformübergreifende Laufzeitumgebung, die es Entwicklern ermöglicht, serverseitige Tools und Anwendungen in JavaScript zu erstellen. Es läuft außerhalb des Browsers, lässt die Browser-spezifischen APIs weg und bietet Unterstützung für Betriebssystem-APIs wie HTTP und Dateisysteme. Node.js bietet viele Vorteile wie z.B. eine gute Leistung, besonders bei skalierbaren und Echtzeitanwendungen. Der Code wird in JavaScript geschrieben, was den Vorteil hat, dass das Frontend und Backend in der selben Sprache geschrieben werden können. Der Node Package Manager (npm) bietet Zugriff auf viele Pakete und ein effizientes Abhängigkeitsmanagement. Node.js ist auch auf vielen Betriebssystemen portabel und wird von vielen Webhosting-Anbietern unterstützt.

Express.js [43] ist ein Framework für Node.js. Es bietet eine einfache API für grundlegende Funktionen von Node und dient als Abstraktionsschicht über dem HTTP-Modul der Node Core API. Express bietet zusätzliche Funktionalitäten für das HTTP-Modul wie die Bearbeitung von Anfragen, aber auch Routing und das Rendern statischer Inhalte. Für die Bearbeitung von Anfragen wird das Middleware Konzept verwendet. Middleware ist eine Funktion, die den Bearbeitungsprozess von Anfragen in mehreren Schritten aufteilt. Diese Funktion kann beliebigen Code ausführen, Änderungen an den Anfrage- und Antwort-Objekten vornehmen, den Anfrage-Antwort-Zyklus beenden oder die nächste Middleware-Funktion im Stapel aufrufen, um die Bearbeitung des Request-Objekts fortzusetzen. Routing sorgt dafür, dass eingehende Anfragen, basierend auf den URL, zu den passenden Teil der Anwendung geleitet werden. Webanwendungen bestehen meistens aus mehreren Seiten, die über bestimmte URLs erreicht werden können, oft auch mit zusätzlichen Parametern wie IDs. Außerdem hat Express eingebaute Funktionen, um CRUD (Create Read Update Delete) Operationen zu bearbeiten. Statische Inhalte sind Inhalte, die der Server dem Client zur Verfügung stellt, ohne dass sie dynamisch generiert werden. Das sind häufig HTML-, CSS- und JavaScript-Dateien, die zur Darstellung von Frontend-Seiten verwendet werden.

### 2.5.3.2 React JS

React [18, 8] ist eine open-source JavaScript Bibliothek, entwickelt von Facebook, die zur Erstellung von interaktiven Webanwendungen dient. Zu den Hauptmerkmalen von React gehört die Verwendung von Komponenten anstelle von Templates. Komponenten kombinieren die in JavaScript geschriebene Logik mit dem HTML Code. Komponenten verwalten ihren Zustand und können zu komplexeren Strukturen zusammengefügt werden. React sorgt

dafür, dass beim Änderungen des Zustands ein Re-Rendering erfolgt. Für ein effizientes Re-Rendering wird ein sogenanntes Virtual-DOM eingesetzt. React verfügt über ein eigenes Event-System um die plattformübergreifende Entwicklung zu verbessern. Das JSX-Format ermöglicht es, HTML Code direkt in den JavaScript Code einzufügen. Außerdem werden Lifecycle Hooks unterstützt die bei bestimmten Ereignissen ausgeführt werden, wie z.B `componentDidMount()` oder `componentWillUnmount()`. React kann sowohl auf Node-Servern gerendert als auch zur Entwicklung von mobilen Apps mit React Native verwendet werden.

### 2.5.3.3 Verschlüsselung der Verbindungen mit TLS

Transport Layer Security (TLS) [49] ist ein Sicherheitsprotokoll, das Datenschutz und Sicherheit bei Kommunikation über das Internet gewährleistet und ist der Nachfolger von den Secure Socket Layer (SSL) Protokoll. Am häufigsten wird es zur Verschlüsselung von Kommunikationen zwischen Webanwendungen und Servern verwendet. TLS wird in Kombination mit HTTP zur Verschlüsselung eingesetzt, wodurch HTTPS [48] entsteht. HTTPS ist der Sicherheitsstandard für die Kommunikation zwischen Webbrowsern und Websites, da es die Daten zwischen beiden verschlüsselt und schützt sind. Moderne Browser zeigen nicht-HTTPS Seiten als potenziell gefährlich an und blockieren einige Funktionen wie z.B den Zugriff auf Mikrofon oder Kamera. TLS erfüllt drei Hauptkriterien: Verschlüsselung, Authentifizierung und Integrität. Dafür werden sogenannte TLS bzw SSL-Zertifikate verwendet. Diese werden von einer Zertifizierungsstelle für den Betreiber der Anwendung ausgestellt und enthalten Informationen wem die Domain gehört und den öffentlichen Schlüssel des Servers.[10]

Ein HTTPS Verbindungsaufbau besteht aus zwei Phasen. Zuerst erfolgt der TCP-Handshake und danach der TLS Handshake. Erst dann können Daten zwischen den beiden Kommunikationspartnern sicher ausgetauscht werden. Wie der TLS-Handshake genau erfolgt hängt von der Art des verwendeten Schlüsselaustausch-Algorithmus und den von beiden Seiten unterstützten Cipher Suites ab. Ein TLS-Handshake mit RSA-Verschlüsselung folgt beispielsweise folgendermaßen ab: Zuerst sendet der Client eine "Client Hello" Nachricht mit unterstützter TLS-Versionen, Verschlüsselungssuiten und einem zufälligen "Client Random". Der Server antwortet mit einer "Server Hello"-Nachricht, die sein SSL-Zertifikat, die gewählte Verschlüsselung und ein "Server Random" enthält. Der Client überprüft das Zertifikat bei der Zertifizierungsstelle und sendet ein, mit den öffentlichen Schlüssel verschlüsseltes, "Premaster Secret" (zufällige Bytefolge). Der Server kann es mit dem privaten Schlüssel entschlüsseln. Beide Seiten erzeugen identische Sitzungsschlüssel aus dem Client Random, Server Random und Premaster Secret. Dann wird von beide Seiten eine mit dem Sitzungsschlüssel verschlüsselte "Fertig" Nachricht gesendet und die verschlüsselte Kommunikation kann beginnen. Abbildung 2.7 zeigt den generellen Ablauf eines TLS-Handshake.[11]

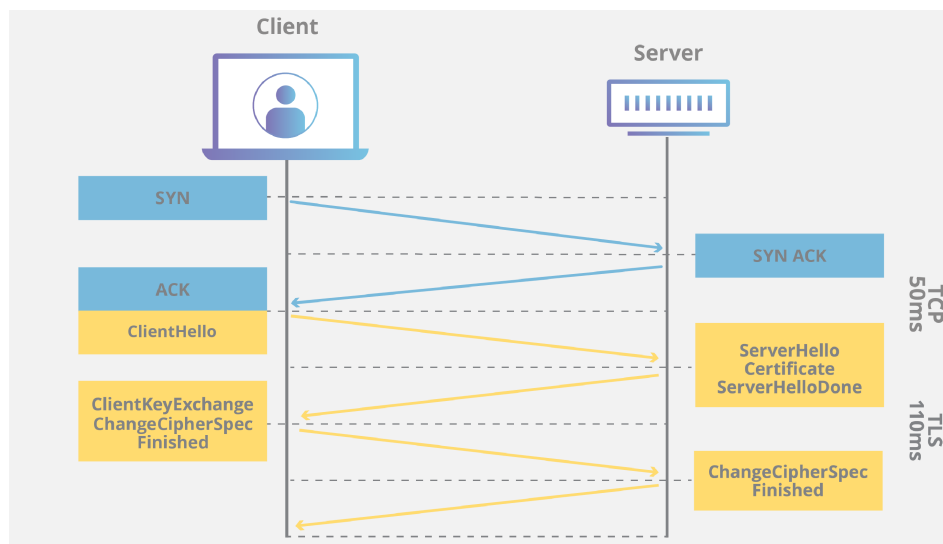


Abbildung 2.7: TLS-Handshake Sequenz

# Kapitel 3

## Design

In diesem Kapitel wird das Design des Projekts entworfen. Der Prozess wird in mehreren Schritten unterteilt. Zuerst wird das Konzept für die Basisanwendung entwickelt, die die grundlegenden Funktionen eines webbasierten Remote-Desktop-Viewer bereitstellt. Als nächstes werden die Lösungen für Audioübertragung und Drucken konzipiert, und abschließend werden alle Komponenten zusammengeführt.

### 3.1 Design der Basis Remote-Desktop Anwendung

Als erstes wird das Design für die Basis-Anwendung entworfen. Sie soll die Grundfunktionalitäten von einem webbasierten Remote-Desktop-Viewer anbieten. Dazu gehört die Bildübertragung, Tastatureingaben sowie Mausinteraktionen. Die Anwendung soll in der Lage sein sich mit Remote-Desktops zu verbinden und über die Protokolle RDP und VNC kommunizieren.

Für den Entwicklungsprozess werden zwei VMs bereitgestellt, eine mit Linux und eine mit Windows. Auf die Linux-VM soll ein VNC- und ein RDP-Server installiert werden, auf der Windows VM soll der bereits vorhandene RDP Server verwendet werden. Wie im Kapitel Kriterien 2.1 beschrieben, ist die Voraussetzung der Apache Guacamole Server (guacd) als Vermittler zu benutzen. Darauf werden die benötigten Client-Plugins für RDP- und VNC-Unterstützung installiert, die die Daten von den VMs erhalten, ins Guacamole Protokoll übersetzen und an der Web-Anwendung weiterleiten. Die Nutzereingaben vom Browser werden ebenfalls zuerst an guacd geleitet, und dann an den VMs.

Guacd alleine ist nicht ausreichend um den Backend abzuschließen. Er ist so konzipiert, dass die Verbindung zusätzlich durch einen Backend Server umgeleitet werden muss. Guacd kommuniziert mit dem Backend über TCP. Das Backend kommuniziert wiederum mit dem Browser über eine WebSocket Verbindung. Aus diesem Grund muss noch ein zusätzlicher (Proxy)-Server implementiert werden. Der Server kann prinzipiell mit jeder Programmiersprache implementiert werden, für die es eine Guacamole-Backend Bibliothek gibt. Für dieses Projekt wird Node.js verwendet. Die Gründe dafür werden im Kapitel 4.2 erläutert.

Schließlich wird auch eine Frontend Komponente benötigt, die den eigentlichen Viewer darstellt. Dafür soll eine einfache Benutzeroberfläche erstellt werden. Der Nutzer soll in der Lage sein zwischen seine Desktops zu wechseln, sie zu steuern und Informationen über die jeweilige Verbindung zu erhalten. Dafür kann prinzipiell jedes JavaScript Web-Framework verwendet werden. Für diese Arbeit wird React verwendet. Um die Guacamole-Sitzung im Browser darzustellen, existiert zurzeit nur die offizielle "guacamole-common-js" [23] Bibliothek von dem Apache Guacamole Projekt. Diese implementiert alles Nötige um den Remote-Desktop im Browser darzustellen und ihn zu steuern. Abbildung 3.1 fasst die beschriebene Architektur der Basisanwendung zusammen.

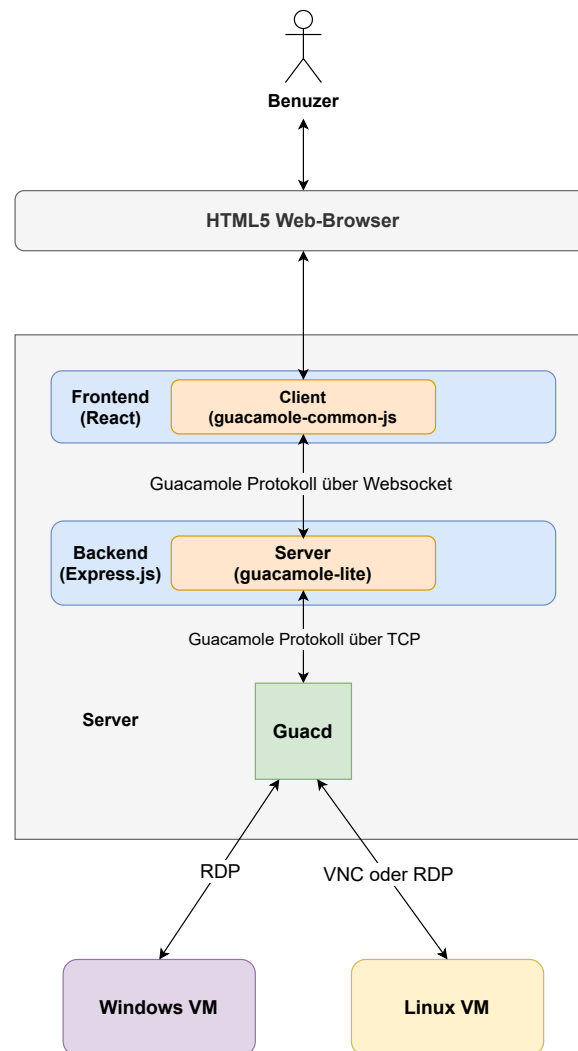


Abbildung 3.1: Architektur der Basis-Anwendung.

## 3.2 Erweiterung um eine Druckoption

Für die Erweiterung um eine Druckoption muss generell zwischen den zwei Protokolle VNC und RDP unterschieden werden. RDP unterstützt die Druckerumleitung, während VNC keine solche Funktionalität bietet. Im Abschnitt 2.3 wurden zwei Lösungsmodelle vorgestellt. Der erste ist den Auftrag direkt an einen physischen Drucker zu senden. Das könnte beispielsweise direkt über das IPP-Protoll erfolgen oder über einer Cloud-Printing Lösung. Allerdings besteht hier das Problem, dass es für einige Nutzer eventuell etwas komplizierter werden kann deren privaten Drucker im Netz freizugeben oder in einer Cloud zu registrieren. Außerdem entstehen für Cloud-Printing Dienste oft Gebühren. Den Druckauftrag über RDP direkt an einen physischen Drucker zu leiten ist zwar möglich, allerdings ist diese Lösung für Desktop basierte Remote-Desktop-Viewer konzipiert. Guacamole ist nicht in der Lage den Auftrag über den Browser an den Drucker zu leiten. Das Drucken über PDF hingegen, vereinfacht das ganze Prozess für den Benutzer. Der benutzer kann einfach auf "Dokument drucken" klicken, und dannach die erzeugte PDF über den Browser herunterladen. Das kann er dann wie gewohnt bei sich lokal Drucken.

Aus diesem Grund wird das Lösungsmodell, Druckauftrag als PDF herunterzuladen, implementiert. Guacamole bietet eine solche Option für RDP-Verbindungen an. Es besteht Möglichkeit den Remote-Desktop mit einen virtuellen Drucker (GhostScript), der von guacd gesteuert wird, zu verbinden und das erzeugte PDF an den Client über das Guacamole-Protokoll zu senden. Das heißt an der Gesamtarchitektur muss nichts geändert werden. Für VNC Verbindungen gibt es keine solche Möglichkeit. Deswegen wird ein eigenes Konzept entwickelt. Die Idee dabei ist einen CUPS-Druckerserver in einen Docker-Container laufen zu lassen. Dabei für jeden Nutzer einen PDF-Drucker aufgesetzt der die Druckaufträge zu PDF umwandelt. Der CUPS Server wird innerhalb des Netzwerks laufen, wo sich die Remote-Desktops befinden. Die PDF Drucker wären somit über IPP erreichbar. Um die erzeugten PDFs an den Nutzer sofort zu schicken, kann ein Postprocessing-Skript verwendet werden, der eine Benachrichtigung, mit dem Namen und Pfad von der PDF-Datei, an den Client schickt. Der Client kann dann das erzeugte PDF über eine Fetch-API herunterladen.

## 3.3 Erweiterung um Audioübertragung

Bei der Audioübertragung muss ebenfalls zwischen RDP- und VNC-Verbindungen unterschieden werden. RDP unterstützt die Umleitung von Audio in beiden Richtungen, VNC hingegen nicht. Die Lösung für Audioübertragung über RDP existiert bereits und funktioniert gut mit geringe Verzögerungen (Siehe Mulfari et al. [39]). Außerdem verläuft das Audio über die selbe Verbindung wie der Rest der Daten. Somit bleiben Bild und Ton synchron. Bei Lösungen, die das Audio über zusätzliche Verbindungen leiten, besteht die Gefahr, dass Bild und Ton asynchron werden können. Aus diesem Grund wird für die RDP-Verbindungen die eingebaute Lösung von Guacamole verwendet. An der Architektur würde sich somit nichts ändern. Alles was gemacht werden muss, ist die Verbindungsparametern anzupassen und die entsprechende Implementierung im Frontend vorzunehmen.

Für VNC-Verbindungen bietet Guacamole eine Lösung für die Audioausgabe. Die VMs können so konfiguriert werden, dass die Audioausgabe über den PulseAudio-Server gestreamt wird. Guacd kann sich mit dem PulseAudio-Server von den VMs verbinden und empfangenen Audiodaten mit den VNC-Daten kombinieren und über das Guacamole-Protokoll an den Client leiten. Der Vorteil dabei ist, dass auch hier keine zusätzliche Verbindung zum Client aufgebaut werden muss. Außerdem werden bei diesem Ansatz keine zusätzlichen Komponenten benötigt, da PulseAudio auf den meisten Linux-Distributionen standardmäßig installiert ist. Aus diesem Grund wird diese Lösung implementiert.

Für die Audioeingabe bei VNC-Verbindungen bleibt keine andere Option als eine zusätzliche Verbindung zu verwenden, da es dafür keine Unterstützung von Guacamole gibt. Zur Option stehen das SPICE-Protokoll, WebRTC oder die Verwendung von den PulseAudio-Server. Das SPICE-Protokoll wäre grundsätzlich eine mögliche Lösung. Allerdings erfordert die Implementierung zusätzliche Komponenten wie einen SPICE-Server, einen Proxy und einen webbasierten SPICE-Client. Um sicherzustellen, dass nur das Audio und nicht der gesamte Desktop übertragen wird, müssten sowohl der Server als auch der Client spezifisch konfiguriert werden. Eine Implementierung wäre eventuell möglich, würde aber Unsicherheiten mit sich bringen. WebRTC wäre ebenfalls eine mögliche Lösung, allerdings ist die Architektur etwas komplizierter, da auch STUN- und TURN-Server benötigt werden und das Audio nicht über die Backend-Komponente verlaufen wird. Außerdem wird ein zusätzlicher Desktop-basierter WebRTC-Client für die VMs benötigt. Die sicherste Lösung ist, die Audioeingabe über den PulseAudio-Server an der VM zu leiten. Der Web-Client kann mithilfe einer Browser-API, die auf das Mikrofon zugreifen kann, das Audio aufnehmen. Die aufgenommenen Audio-Chunks werden dann über eine zusätzliche WebSocket-Verbindung an den Proxy-Server gesendet. Dieser kann sie an den PulseAudio-Server von der VM schicken. Eine direkte Verbindung von dem Browser zu PulseAudio ist nicht möglich, da der Browser keine TCP-Verbindungen unterstützt und keine Bibliotheken für die direkte Kommunikation gefunden werden konnten.

## 3.4 Gesamtarchitektur

Kombiniert man den Design von der Basisanwendung, die Druckerfunktionalität und Audioübertragung entsteht die in Abbildung 3.2 abgebildete Architektur. Für RDP-Verbindungen ändert sich an der Architektur nichts, da alles über die selbe Verbindung verläuft. Für die VNC-Verbindungen müssen noch zusätzliche Komponenten entwickelt und konfiguriert werden. Für die Audioeingabe muss ein Client im Frontend entwickelt werden, der das Audio aufnimmt, sowie eine WebSocket-API, die die Aufnahmen an den PulseAudio-Server von den Remote-Desktop leitet. Für das Drucken wird ebenfalls ein Client im Frontend benötigt sowie eine WebSocket-API im Backend und die CUPS-Server-Komponente, wo die PDF-Drucker laufen.

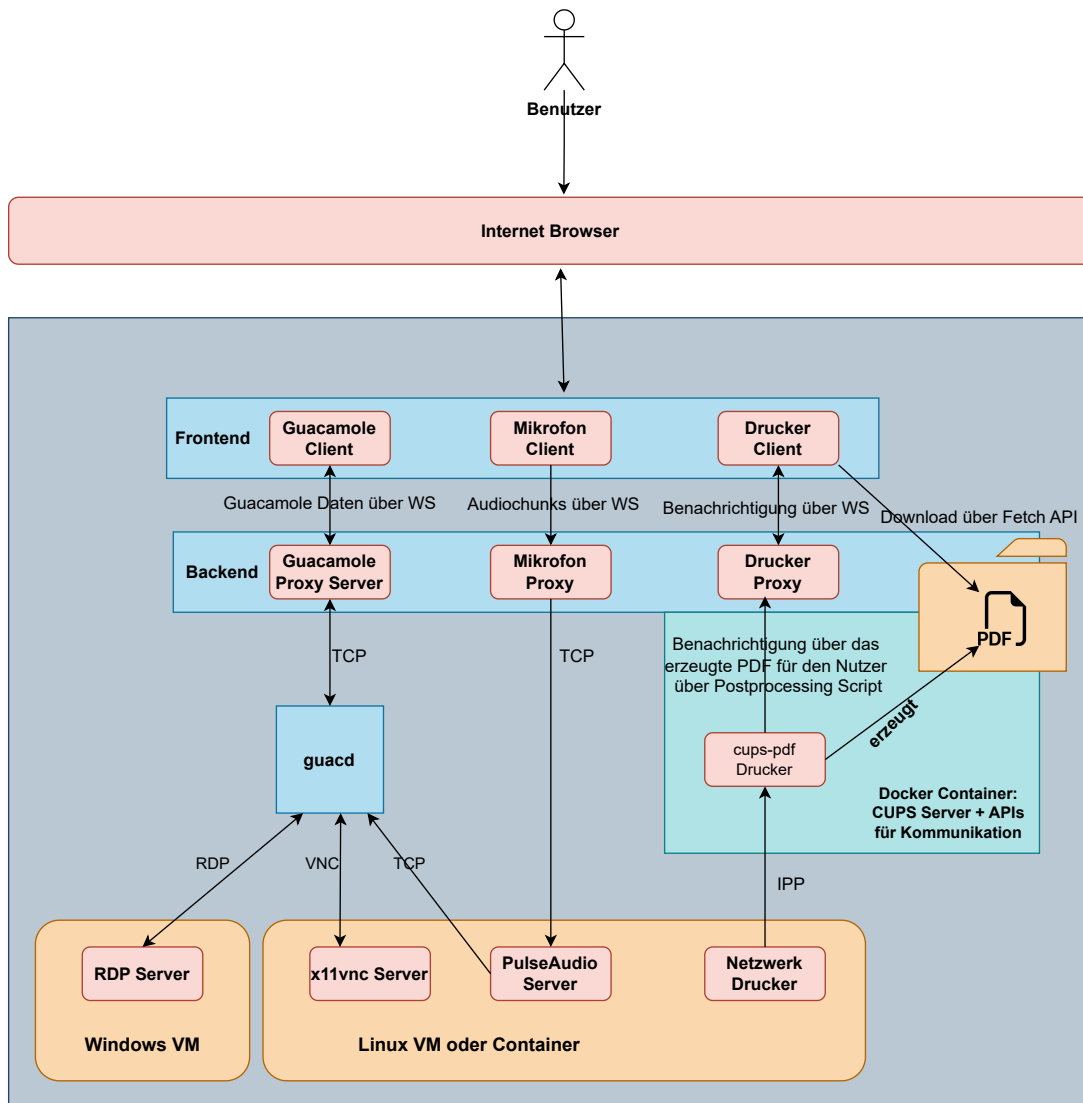


Abbildung 3.2: Erweiterung für VNC Verbindungen



# Kapitel 4

## Implementierung

In diesem Kapitel wird die Implementierung des beschriebenen Designs erläutert. Zuerst wird die Umgebung aufgesetzt. Das umfasst die Installation von Proxmox, die benötigten VMs und die Bereitstellung von guacd. Als nächstes wird die Basis-Anwendung erzeugt, womit der Benutzer die Desktops steuern kann. Wenn die Basis-Anwendung implementiert ist, werden die Audio- und Druck-Funktionen integriert. Zum Schluss werden die Verbindungen mit TLS gesichert und die ganze Anwendung wird als Docker-Container bereitgestellt.

### 4.1 Entwicklungsumgebung

Der erste Schritt ist das Aufsetzen einer Entwicklungsumgebung für die Implementierung des Projekts. Die Umgebung des Projekts besteht aus einer PVE Instanz, wo die zu steuernden VMs laufen und eine Guacamole-Server (guacd) Instanz. Proxmox kann entweder als VM z.B in VirtualBox oder als Host-System auf einem Gerät installiert werden. Wenn es als VM installiert wird, muss beachtet werden, dass die VM genug Ressourcen zugewiesen bekommt, da innerhalb weitere VMs laufen werden. In dieser Projektumgebung wird Proxmox-VE als Hauptbetriebssystem auf einen ASUS S56C Laptop installiert werden der als Server dienen wird. Für die Installation wurde ein bootfähiges Laufwerk auf einem USB-Stick mithilfe des Programms RUFUS erstellt. Die Installation erfolgt über ein grafisches Installationsmenü. Nach der Installation ist PVE sofort betriebsbereit. Es kann über die Kommandozeile gesteuert werden, allerdings gibt es eine webbasierte Benutzeroberfläche (Abb. 4.1), die z.B über den Arbeit-PC benutzt werden kann. Die Web-Oberfläche ist unter Port 8006 erreichbar.

Nachdem Proxmox erfolgreich installiert wurde, müssen die benötigten VMs darauf installiert werden. Zuerst wurde eine Windows 10 Pro VM installiert. Es ist wichtig die Pro Version zu wählen, da auf der Home-Version die Fernsteuerung nicht aktiviert werden kann ist. Nach der Installation, muss die Option zur Fernsteuerung in den Windows-Einstellungen aktiviert werden. Der RDP-Server und Client sind beim Windows 10 Pro standardmäßig installiert und nach der Aktivierung sofort einsatzbereit. Um die Verbindung zu testen, kann beispielsweise der Remote-Desktop-Viewer, der auch für die Home-Version verfügbar ist, von einem anderen Windows Gerät benutzt werden.

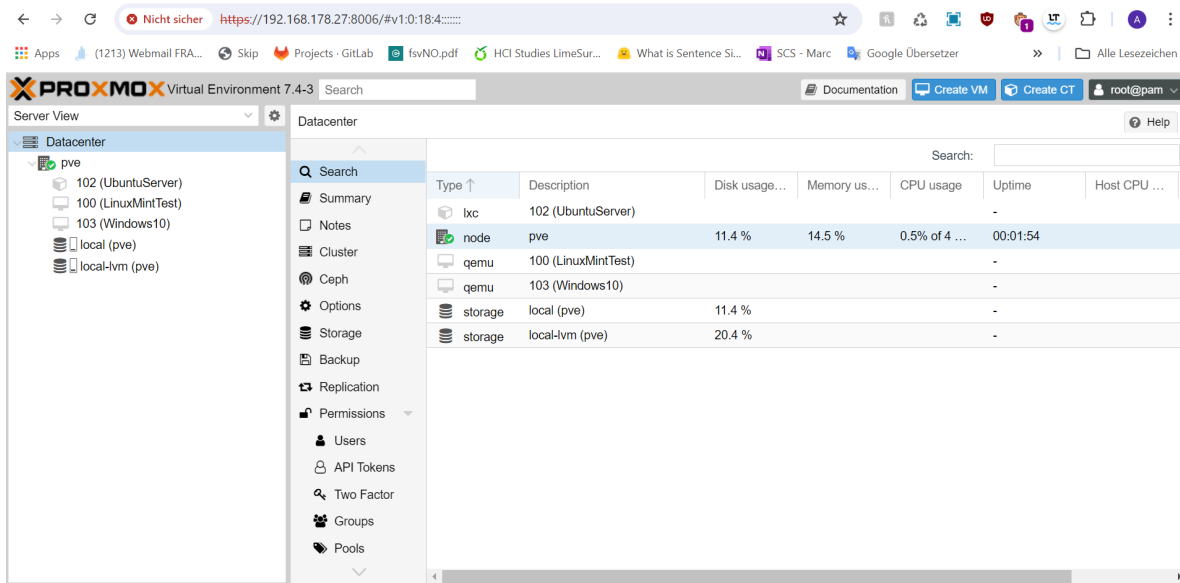


Abbildung 4.1: Web-basierte Benutzeroberfläche von Proxmox-VE.

Als nächstes wurde eine Linux-VM installiert. Dafür wurde die Mint Distribution gewählt. Beim Linux ist keine Remote-Steuerung standardmäßig installiert, deswegen müssen VNC- oder RDP-Servern zusätzlich installiert werden. Es wurde ein x11vnc Server über apt installiert. Um die Verbindungen abzusichern, wurde ein Passwort mit dem Befehl `x11vnc -storepasswd` erstellt. Damit der Server immer läuft, auch nach einem Neustart, muss der x11vnc Service entsprechend angepasst werden. Dafür muss die Datei `/lib/systemd/system/x11vnc.service` wie im Listing 4 definiert werden. Um auch RDP-Verbindungen für Linux zu ermöglichen wurde noch zusätzlich der xRDP-Server aufgesetzt. Dieser kann ebenfalls über apt installiert werden und ist sofort nach der Installation einsatzbereit. Seine Funktionalitäten sind jedoch etwas begrenzter als die von den RDP-Server von Windows.

Für das Aufsetzen von guacd gibt es zwei Optionen. Die Erste ist es die Komponente lokal zu bauen. Dazu müssen die nötigen sowie optionale Abhängigkeiten installiert werden. Die zweite Option ist es ihn als Docker-Container auszuführen. Dafür stellt das Guacamole-Projekt einen Docker-Image [16] zur Verfügung, wo alles bereits installiert ist, wie z.B Unterstützung für VNC- und RDP-Verbindungen, PulseAudio, GhostScript und Andere. Für das Projekt wurde die Docker-Lösung verwendet. Listing 5 zeigt wie das guacd-Image aus Dockerhub geladen und daraus eine Container-Instanz erzeugt werden kann. Dabei ist es wichtig den Port 4822 nach außen zu öffnen. Mit der Umgebungsvariable `GUACD_LOG_LEVEL` kann der Log-Level geändert werden, was den Debug-Prozess erleichtert.

```
1 [Unit]
2 Description=x11vnc service
3 After=display-manager.service network.target syslog.target
4
5 [Service]
6 Type=simple
7 ExecStart=/usr/bin/x11vnc -forever -display :0 -auth guess \
8     -passwd <PASSWORD>
9 ExecStop=/usr/bin/killall x11vnc
10 Restart=on-failure
11
12 [Install]
13 WantedBy=multi-user.target
```

Listing 4: x11vnc.service Konfiguration

```
1 docker pull guacamole/guacd
2 docker run -d -p 4822:4822 --name guacd guacamole/guacd
```

Listing 5: Ausführen von einem guacd-Container

## 4.2 Implementierung der Basisanwendung

Als nächstes wurde die Web-Anwendung implementiert, womit Nutzer auf deren Desktops zugreifen können. Das Guacamole Projekt bietet bereits eine solche Anwendung die den Zugriff ermöglicht, Authentifizierung durchführt und Sitzungen verwaltet. Das Ziel dieses Projekts ist es aber den Remote-Desktop in eine eigene Anwendung zu integrieren und um neue Funktionalitäten zu erweitern, die von Guacamole nicht zur Verfügung gestellt werden. Deswegen wird der offizielle Guacamole-Client nicht für die Implementierung verwendet. Es bietet sich aber an, ihn für Testzwecke zu benutzen. Deswegen wurde er dennoch für Testzwecke installiert. Er kann lokal installiert werden, es ist aber deutlich einfacher ihn als Docker-Container laufen zu lassen. Dafür gibt es eine Anleitung [19] wie es gemacht werden kann. Es muss eine MySQL Datenbank für die Nutzerauthentifizierung als Container aufgesetzt und konfiguriert werden. Danach müssen noch die Containern mit den Guacamole Server (guacd) und den Guacamole-Client erstellt und gestartet werden. Für alle der Komponenten gibt es offizielle Images, die aus Dockerhub geladen werden können. Nachdem alles aufgesetzt ist kann die webbasierte Benutzeroberfläche auf der Adresse `http://HOST_IP/guacamole` erreicht werden. Unter Einstellungen können neue Verbindungen hinzugefügt werden. Dafür werden Parameter wie die IP-Adresse des Desktops, Nutzernamen, Passwort und Verbindungstyp benötigt. Auf Abbildung 4.2 ist der Dashboard

von der offiziellen Webanwendung zu sehen. Dort können erstellte Verbindungen ausgewählt werden und der Desktop kann mit allen eingebauten Funktionalitäten, die Guacamole anbietet, gesteuert werden werden.

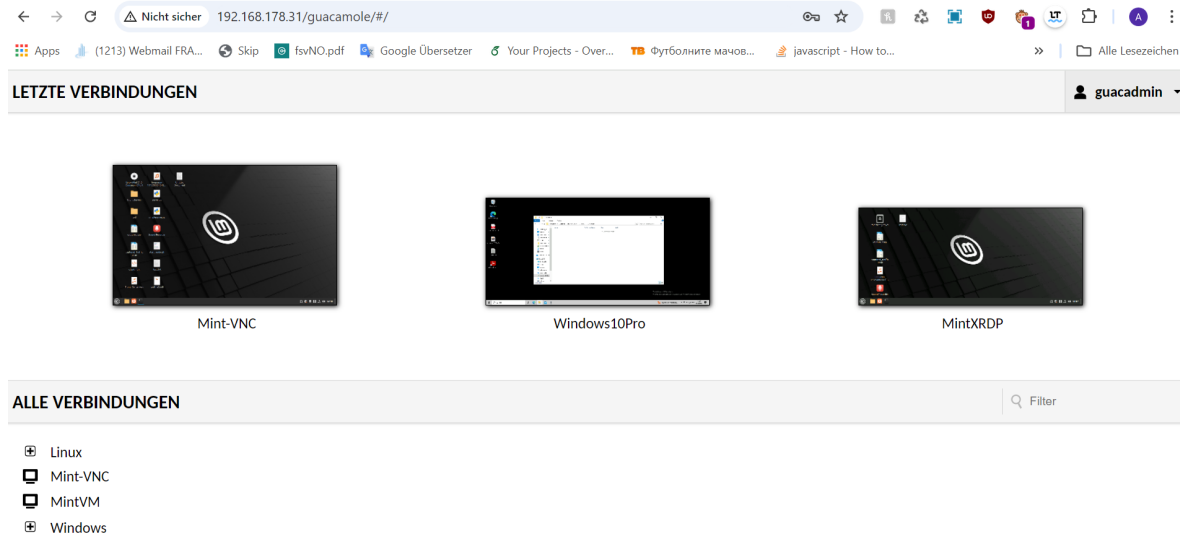


Abbildung 4.2: Web-basierte Benutzeroberfläche von Apache Guacamole

Um den Remote-Desktop-Viewer in eine eigene Anwendung zu integrieren, wird die Guacamole-Client Komponente nicht benötigt. Was benötigt wird, ist `guacd`, ein Proxy-Server, der die Guacamole-Daten an dem Webbrowser weiterleitet und eine Frontend-Komponente, die den Desktop anzeigt. Alle benötigten Bibliotheken sind frei verfügbar.

Als erstes wurde das Backend, bzw. der Proxy-Server, implementiert. Es soll als Vermittler zwischen `guacd` und das Frontend zu dienen, da das Frontend nicht direkt mit `guacd` kommunizieren kann. Es gibt mehrere Möglichkeiten einen solchen Server zu implementieren. Er kann theoretisch von Grund aus selbst implementiert werden, allerdings existieren mehrere Bibliotheken, die ihn bereits implementieren. Dazu gehören beispielsweise `pyguacamole` [47] für Python, `guacamole-sharp` [55] für C#, `guacamole-common` [22] für Java oder `guacamole-lite` [44] für Node.js. Davon ist nur `guacamole-common` Teil des Guacamole Projekts, bei den anderen handelt es sich um externe Bibliotheken. Für diese Arbeit wird `guacamole-lite` benutzt. Es ist leicht zu benutzen, lässt sich einfach in Projekte integrieren und bietet alle benötigten Funktionalitäten für dieses Projekt. Von den externen Bibliotheken hat `guacamole-lite` den aktuellsten Code und die Git-Repository wird noch verwaltet. Da es sich dabei um einen JavaScript Bibliothek handelt, wird für das Backend das Express.js Framework für Node.js benutzt.

Die zentrale Komponente der Bibliothek `guacamole-lite` ist die Klasse `Server`. Dabei handelt es sich um einen Websocket-Server, der mit den Web-Clients kommuniziert und Sitzungen verwaltet. Für jede neue Sitzung wird einen `ClientConnection` Objekt erstellt. Dieser verwendet wiederum einen `GuacdClient` Objekt, um mit `guacd` zu kommunizieren.

ClientConnection leitet die empfangenen Nachrichten von guacd an den Web-Client weiter und andersherum. Um Sicherheit zu gewährleisten verwendet guacamole-lite folgendes Konzept. Der Web-Client schickt beim Verbindungsaufbau einen verschlüsselten Token an den Server. Dieser Token enthält alle Parameter, die benötigt werden um den Handshake auszuführen und sich mit dem Remote-Desktop zu verbinden, wie z.B Adresse, Benutzername oder Passwort. Um den Token zu entschlüsseln wird die Klasse Crypt verwendet. Wie der Web-Client den Token bekommt, ist den Entwickler selbst überlassen. Die Bibliothek enthält einige Beispiel-Skripte um solche Tokens zu generieren. Listing 6 zeigt, wie ein einfacher guacamole-lite Server erstellt werden kann. Dabei wird zuerst ein HTTP Server mithilfe von Express erstellt. Als nächstes wird eine Guacamole-Server Instanz erstellt. Diese benötigt als Parameter die Client-Konfigurationen, guacd-Konfigurationen und den HTTP Server. Es gibt auch zusätzliche Konfigurationen, diese werden aber nicht gebraucht. Der Server kann auf einen beliebigen Port gestartet werden, wie z.B 8080.

```
1  const GuacamoleLite = require('guacamole-lite');
2  const express = require('express');
3  const http = require('http');
4
5  const app = express();
6
7  const server = http.createServer(app);
8
9  const guacdOptions = {
10     port: 4822 // port of guacd
11 };
12
13 const clientOptions = {
14     crypt: {
15         cypher: 'AES-256-CBC',
16         key: 'MySuperSecretKeyForParamsToken12'
17     }
18 };
19
20 const guacServer = new GuacamoleLite(
21     {server},
22     guacdOptions,
23     clientOptions
24 );
25
26 server.listen(8080);
```

Listing 6: Beispiel für einen einfachen guacamole-lite Server

Als nächstes wurde das Frontend entwickelt. Dafür wurde das Framework React für JavaScript verwendet. Das Ziel dabei ist es nicht eine komplexe Anwendung zu erstellen, sondern nur das nötigsten Funktionalitäten bereitzustellen, um sich mit einem Desktop zu verbinden und ihn fernzusteuern. Als erstes wurde ein Verbindungsformular (Abbildung 4.3) implementiert. Dort kann der Nutzer seine Verbindungsdaten eintragen um sich mit dem Desktop zu verbinden. Zu den Verbindungsdaten gehören der Token, Verbindungstyp, IP-Adresse und für VNC-Verbindungen ein Gastname da die Anwendung über keine Benutzerkonten verfügt. Um Tokens zu generieren können die Skripten benutzt werden, die von guacamole-lite zur Verfügung stehen. Dort können die verschiedenen Verbindungsparametern angepasst werden.

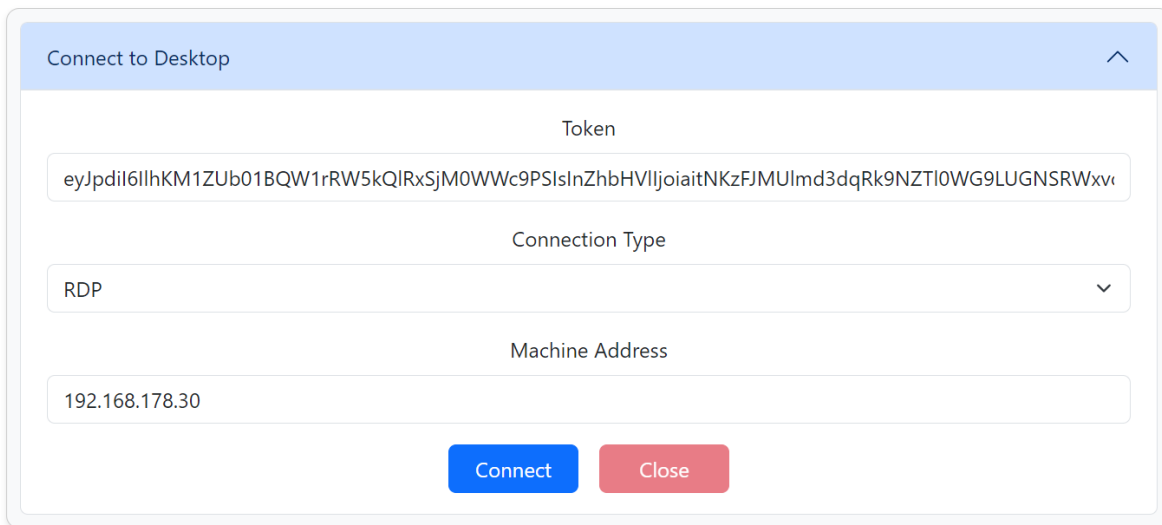


Abbildung 4.3: Formular für die Herstellung von Verbindungen

Danach wurde der Hauptteil des Frontends, der eigentliche Desktop-Viewer, implementiert. Dafür wurde die React-Komponente `GuacamoleStage.jsx` erstellt. Sie ist dafür verantwortlich, mithilfe des angegebenen Tokens eine Websocket Verbindung zu dem Proxy-Server aufzubauen und den Desktop im Browser darzustellen. Um eine Guacamole-Verbindung aufzubauen, wurde die Klasse `Guacamole.WebSocketTunnel` verwendet. Die Klasse `Guacamole.Client` ist der eigentliche Client, der die Steuerung der Sitzung übernimmt, indem er die Verbindung über den WebSocket-Tunnel nutzt. Der Client verarbeitet die Daten, die vom Server über den Tunnel kommen wie z.B. Graphiken und stellt sie im Browser dar. Außerdem verwaltet er die Benutzerinteraktionen wie Tastatureingaben und Mausbewegungen und sendet sie an den Server über den Tunnel. Listing 7 zeigt wie eine einfache Verbindung aussehen kann. Dabei wird eine Verbindung zu dem guacamole-lite Server aufgebaut der auf Port 8080 läuft. Mit der Funktion `client.getDisplay().getElement()` kann eine Referenz zu dem Display erstellt werden, die dann im Browser abgebildet werden kann.

```
1 // Code ...
2
3 const tunnel = new Guacamole.WebSocketTunnel(`ws://localhost:8080/`);
4 const client = new Guacamole.Client(tunnel);
5 this.myRef.current.appendChild(client.getDisplay().getElement());
6 this.client.connect('token='+this.token);
7
8 // Code ...
9
10 render() {
11     return <div ref={this.myRef} />;
12 }
```

Listing 7: Verbindungsaufbau zu dem Guacamole-Server

Um Maus und Tastatur Interaktionen an den Server zu senden, gibt es die entsprechende Klassen `Guacamole.Mouse` und `Guacamole.Keyboard`. Listing 8 zeigt, wie das ganze im Code umgesetzt werden kann. Dabei werden alle Mausbewegungen und Tastatureingaben registriert und mithilfe des Guacamole-Clients an den Server geschickt.

```
1 // Mouse Events
2 const mouse = new Guacamole.Mouse(client.getDisplay().getElement());
3 mouse.onEach(
4     ['mousedown', 'mouseup', 'mousemove'],
5     function sendMouseEvent(e) {
6         client.sendMouseState(e.state);
7     }
8 );
9
10 // Keyboard Events
11 const keyboard = new Guacamole.Keyboard(document);
12 keyboard.onkeydown = function (keysym) {
13     client.sendKeyEvent(1, keysym);
14 };
15 keyboard.onkeyup = function (keysym) {
16     client.sendKeyEvent(0, keysym);
17 };
```

Listing 8: Implementierung von Listener für Maus- und Tastatur-Eingabe

Somit wurde eine Anwendung entwickelt, die als Basis für die Entwicklung von einer Druckreption und die Integrierung von Audioübertragung dienen kann. Die Basisanwendung ist minimalistisch. Funktionalitäten wie Authentifizierung, Navigationsmenü oder Änderung der

## KAPITEL 4. IMPLEMENTIERUNG

Bildschirmgröße wurden nicht implementiert, da sich die Anwendung hauptsächlich auf den eigentlichen Viewer fokussiert. Abbildungen 4.4 und 4.5 zeigen jeweils die bisher entwickelte Benutzeroberfläche. Sie besteht aus einer ein ausklappbares Menü, wo sich der Verbindungsformular befindet, ein ausklappbares Toolbar und der eigentliche Desktop-Viewer, der zu bedienen ist. Abbildung 4.4 zeigt eine VNC-Verbindung zu einem Linux Mint Desktop und Abbildung 4.5 zeigt eine RDP-Verbindung zu einem Windows 10 Desktop. Beide Desktops können ganz normal benutzt werden. Mausbewegungen und Tastatureingaben werden übertragen und das Bild wird ständig aktualisiert, sodass keine spürbare Verzögerungen auftreten.

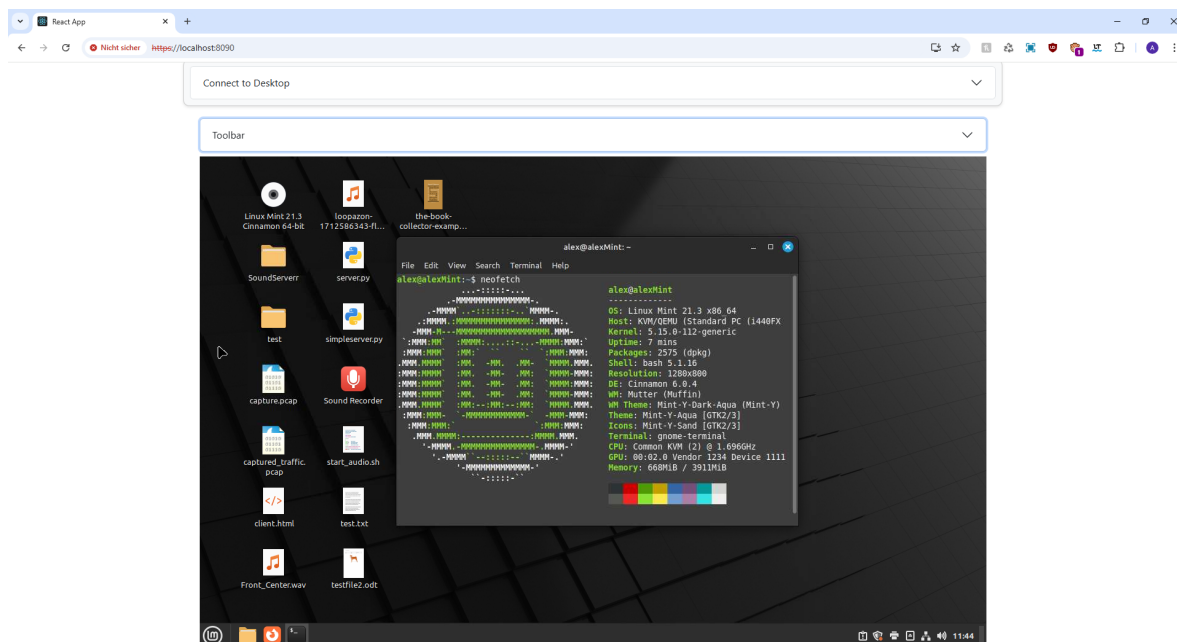


Abbildung 4.4: Linux Mint Verbindung über VNC

Für alle guacd-Verbindungen werden Logs erstellt. Listing 9 zeigt beispielsweise welche Ereignisse bei einer erfolgreichen RDP-Verbindung geloggt werden. Die Logs zeigen, dass zuerst ein RDP-Client mit der ID "\$ff751b00-4300-431e-875a-a9208b6d27c8" erstellt wurde. Der Sicherheitsmodus, der für die Verbindung benutzt wird, ist "Negotiate (ANY)" und es wird keine Resize-Methode angewendet. Zusätzlichb wird angezeigt, dass keine Zeilenenden-Normalisierung für die Zwischenablage spezifiziert sind und, dass die Keymaps „base“ und „en-us-qwerty“ geladen wurden. Danach war der Benutzer erfolgreich angemeldet. Für VNC Verbindungen werden ebenfalls ähnliche Ereignisse gelogt.



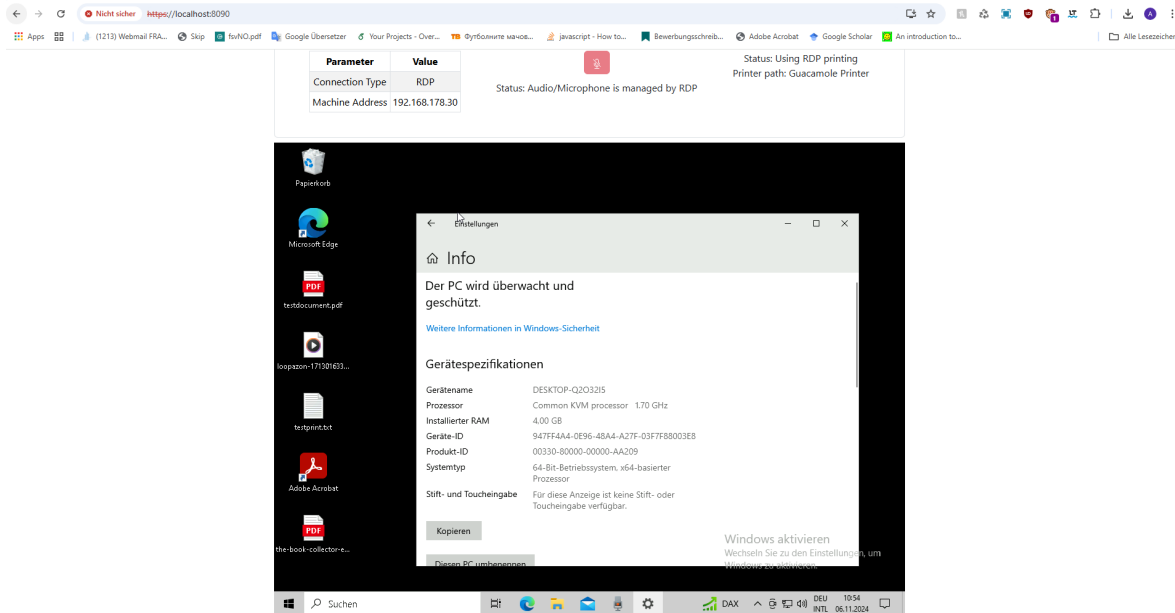


Abbildung 4.5: Windows 10 Verbindung über RDP

```

1 guacd[1]: INFO: Creating new client for protocol "rdp"
2 guacd[1]: INFO: Connection ID is "$ff751b00-4300-431e-875a-a9208b6d27c8"
3 guacd[880]: INFO: Security mode: Negotiate (ANY)
4 guacd[880]: INFO: Resize method: none
5 guacd[880]: INFO: No clipboard line-ending normalization specified.
6 Defaulting to preserving the format of all line endings.
7 guacd[880]: INFO: User "@fd12ec88-93fa-4e28-a813-2718c81e39af" joined
8 connection "$ff751b00-4300-431e-875a-a9208b6d27c8" (1 users now present)
9 guacd[880]: INFO: Loading keymap "base"
10 guacd[880]: INFO: Loading keymap "en-us-qwerty"
11 guacd[880]: INFO: Connected to RDPDR 1.13 as client 0x0004
12 guacd[880]: INFO: Connected to RDPDR 1.13 as client 0x0001
13 guacd[880]: INFO: RDPDR user logged on

```

Listing 9: Guacd-Logs bei einer RDP-Verbindung

### 4.3 Implementierung der Druckeroption

Nachdem die Basisanwendung entwickelt wurde, kann mit der Entwicklung von der Druckeroption angefangen werden. Dabei wird es, wie im vorherigem Kapitel beschrieben, zwei unterschiedliche Lösungen geben: eine für RDP und eine für VNC-Verbindungen. Zuerst wird die Lösung für die RDP-Verbindungen implementiert. RDP unterstützt die Umleitung

des Druckers und Guacamole bietet die Möglichkeit den Druckauftrag als PDF an den Client weiterzuleiten. Um das ganze in der Basis-Anwendung zu integrieren, muss zuerst die Druckoption aktiviert werden. Dafür muss der Verbindungsparameter `enable-printing` auf `true` gesetzt werden. Somit erstellt `guacd` einen virtuellen Drucker für die entsprechende Sitzung und verbindet damit den RDP-Server. Die Voraussetzung dafür ist, dass Ghostscript installiert ist und `guacd` den Zugriff darauf hat. Das ist bereits vorhanden in den offiziellen Docker-Image für `guacd`. Mit dem Parameter `printer-name` kann der Name des Druckers optional geändert werden. Der standardmäßigen Name ist "Guacamole Printer". Zu implementieren bleibt das Entgegennehmen der PDF-Datei im Frontend. Dafür kann die Callback-Funktion `onfile` von der Klasse `Guacamole.Client` benutzt werden. Listing 10 zeigt wie die Funktion definiert wurde. Als Parameter werden Stream, Dateityp und Dateiname übergeben. Die Übertragung der Datei erfolgt in einzelnen Chunks über den Stream. Die einzelnen Chunks müssen gesammelt werden und zum Schluss zu einen Blob zusammengefügt werden. Die Zusammengesetzte Datei wird schließlich über den Browser heruntergeladen. Wichtig dabei ist, dass der Stream immer eine Bestätigung an den Guacamole-Server zurückschicken muss, wenn er etwas empfängt, da der Server auf diese wartet um den nächsten Chunk zu schicken.

Um die implementierte Lösung zu benutzen muss beim Drucken, der "Guacamole Printer" (Abb. 4.6) ausgewählt werden. Schaut man sich die `guacd`-Logs an (Abb. 4.7), sieht man, dass ein Druckauftrag und ein PDF Filter Prozess erstellt wurden. Dannach wurde GhostScript ausgeführt und schließlich wurde der Auftrag beendet.

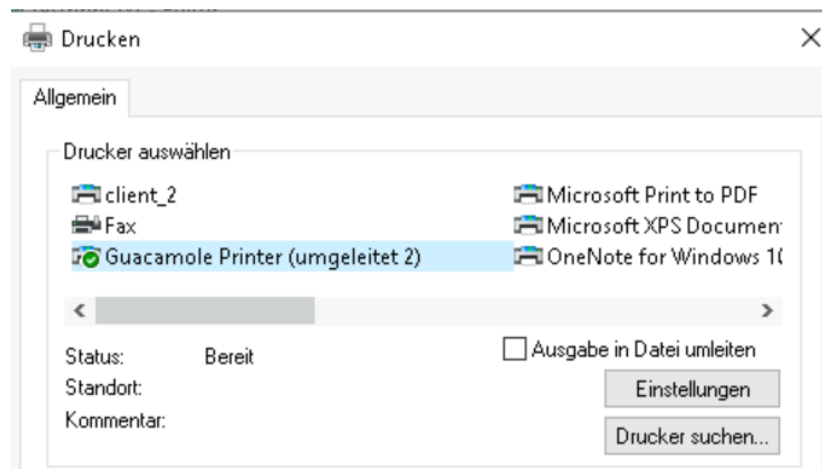


Abbildung 4.6: Guacamole Drucker unter Windows

```

guacd[848]: INFO:      Connected to RDPDR 1115 as client 0x0000
guacd[848]: INFO:      RDPDR user logged on
guacd[848]: INFO:      Print job created
guacd[848]: INFO:      Created PDF filter process PID=867
guacd[867]: INFO:      Running gs
guacd[848]: INFO:      Print job closed
  
```

Abbildung 4.7: Guacd-Logs beim Drucken mit dem Guacamole-Printer

```
1 downloadFile = (stream, mimetype, filename) => {
2   // Remove duplicate .pdf
3   if(filename.endsWith('.pdf.pdf')){
4     filename = filename.slice(0, -4)
5   }
6
7   let dataChunks = [];
8   stream.sendAck('OK', Guacamole.Status.Code.SUCCESS);
9
10  // Collect all chunks
11  stream.onblob = (data64) => {
12    // Decode base64 data chunk
13    let byteCharacters = atob(data64);
14    let byteNumbers = new Array(byteCharacters.length);
15    for (let i = 0; i < byteCharacters.length; i++) {
16      byteNumbers[i] = byteCharacters.charCodeAt(i);
17    }
18    let byteArray = new Uint8Array(byteNumbers);
19    dataChunks.push(byteArray);
20
21    stream.sendAck('OK', Guacamole.Status.Code.SUCCESS);
22  };
23
24  // Connect the chunks to a Blob and download the file
25  stream.onend = () => {
26    stream.sendAck('OK', Guacamole.Status.Code.SUCCESS);
27
28    const blob = new Blob(dataChunks, { type: mimetype });
29
30    const a = document.createElement('a');
31    a.href = URL.createObjectURL(blob);
32    a.download = filename;
33    document.body.appendChild(a);
34    a.click();
35    document.body.removeChild(a);
36  };
37 }
```

Listing 10: Funktion zum Herunterladen von Dateien über das Guacamole-Protokoll

Als nächstes wird die Druckoption für VNC Verbindungen implementiert. Wie im Kapitel Design erklärt, muss hierfür eine eigene Lösung implementiert werden, da VNC und Guacamole keine Lösung dafür anbieten. Die zu implementierende Lösung ist, einen CUPS-Server aufzusetzen, der für jeden Benutzer einen eigenen PDF-Drucker verwaltet. Jeder Nutzer bekommt somit einen Drucker zugewiesen, der auf den Remote-VM als IPP-Drucker hinzugefügt werden kann. Die Aufträge werden an den CUPS-Server gesendet und danach an den Web-Client als PDF Dateien weitergeleitet.

Zunächst wurde der CUPS-Server als Docker-Container aufgesetzt. Dafür wurde ein Dockerfile angelegt, wo alles Nötige definiert wurde (Datei: `PrintServiceVNC/Dockerfile`). Als Basis-Image wurde Ubuntu benutzt. CUPS wurde standardmäßig über apt installiert. Damit der Server auch von außen erreichbar ist, wurden in der Konfigurationsdatei `/etc/cups/cupsd.conf` die ACLs wie im Listing 11 geändert. Somit werden alle Verbindungen aus dem lokalen Netzwerk akzeptiert. Das ist zwar keine sichere Lösung, da es sich aber um ein geschlossenes Netzwerk handelt und die Adressen nicht öffentlich sind, stellt es erstmals kein Problem dar. Um auf den Admin-Seiten zuzugreifen, wird das Root-Passwort benötigt. Die webbasierte Benutzeroberfläche ist auf Port 631 zu erreichen. Dort können Drucker, Warteschlangen und Aufträge verwaltet werden. Alternativ kann CUPS auch über die Kommandozeile benutzt werden. Damit die Konfigurationen nicht jedes Mal bei der Image-Erstellung gemacht werden müssen, wurde die angepasste Konfigurationsdatei in dem Ordner von den Dockerfile hinterlegt und in das Image kopiert.

```
1 # Restrict access to the server...
2 <Location />
3     Order allow,deny
4     Allow from all
5 </Location>
6
7 # Restrict access to the admin pages...
8 <Location /admin>
9     Order allow,deny
10    Allow from all
11 </Location>
```

Listing 11: Konfiguration der ACLs

Der nächste Schritt ist Drucker für die Benutzer der Anwendung hinzuzufügen. Da es sich dabei um PDF-Drucker handelt, wurde dafür CUPS-PDF [60] benutzt. Dieser wandelt die Aufträge zu PDF um und legt sie in einen angegebenen Ordner ab. CUPS-PDF kann über apt installiert werden. Neue Drucker können manuell über die webbasierte Benutzeroberfläche hinzugefügt und im Netz freigegeben werden. Damit das Prozess automatisiert wird, wird die Kommandozeile dafür verwendet. Dafür wurde ein Skript (Datei: `VNC.Print.Service/scripts/create_printer.sh`) geschrieben, der neue Dru-

cker erstellen kann. Der Skript funktioniert folgendermaßen: Zuerst wird einen Ablage-Ordner für den angegebenen Benutzer angelegt, falls er nicht bereits existiert, wie z.B. `/usr/src/files/user1`. Dort sollen alle erzeugte PDFs für diesen Nutzer landen. Als nächstes wird eine Konfigurationsdatei für seinen PDF-Drucker angelegt. Es ist wichtig, dass jeder CUPS-PDF Drucker eine eigene Konfigurationsdatei bekommt, da es dort definiert wird wo die erzeugten PDFs landen. Listing 12 zeigt wie der Ablageort für die erzeugten PDF-Dateien definiert werden kann. Die Zeile `Out` definiert den Ablage-Ordner für Benutzer des Systems (der Docker Container in diesen Fall) und `AnonDirName` für alle anderen. Da die Druckaufträge von anderen Systemen (VMs) kommen werden, reicht es aus nur den `AnonDirName` Pfad anzugeben. Die Konfigurationsdateien werden im Ordner `/etc/cups` abgelegt und sollen folgendermaßen benannt werden: `cups-pdf-<Benutzername>.conf`.

```
1 Out <Your_Output_Directory>
2 AnonDirName <Your_Output_Directory>
```

Listing 12: Konfigurierung des Ablageorts von PDF-Dateien, erzeugt von CUPS-PDF

Dann wird der eigentliche Drucker hinzugefügt, nachdem es überprüft wird ob er nicht bereits existiert. Listing 13 zeigt wie ein Drucker hinzugefügt werden kann. Dabei ist `NAME` der Name des Benutzers. Mit dem ersten Befehl wird der Drucker hinzugefügt, der zweite Befehl sorgt dafür, dass der Drucker Aufträge akzeptiert und mit dem Dritten wird er ins Netz freigegeben [30]. Nachdem der Skript ausgeführt wird kann in der CUPS-Web-Benutzeroberfläche überprüft werden, ob der Drucker erfolgreich erstellt wurde (Abbildung 4.8). Die IPP-Adresse des Druckers lautet `ipp://<Hostadresse>:631/printers/<Benutzername>`.

```
1 lpadmin -p "$NAME" -v "cups-pdf:/$NAME" -E -P
   ↪ /usr/share/ppd/cups-pdf/CUPS-PDF_opt.ppd
2 cupsaccept "$NAME"
3 lpadmin -p "$NAME" -o printer-is-shared=true
```

Listing 13: Hinzufügen von CUPS-PDF Drucker

Der nächste Schritt ist das Ganze in der Basisanwendung zu integrieren. Die Anwendung verfügt über keine Registrierung und somit gibt es keine Benutzernamen. Damit aber die Drucker voneinander unterschieden werden können, wurde für die VNC-Verbindungen das Formularfeld `Gastname` hinzugefügt. Unter dieser Name wird für den Benutzer einen Drucker erzeugt. Diese Lösung ist nicht sicher, da mehrere Nutzer denselben `Gastname` angeben könnten, würde aber für Testzwecke ausreichen. Die Kommunikation mit dem CUPS-Server erfolgt über eine Websocket Verbindung. Dafür läuft neben CUPS auch einen Express Server in den Docker-Container. Da der Browser aus Sicherheitsgründe nicht direkt mit dem Container kommunizieren soll, wird die Verbindung über das Backend zwischen-geleitet, wo eine Authentifizierung stattfinden kann. Sobald sich der Client mit dem Container verbindet,

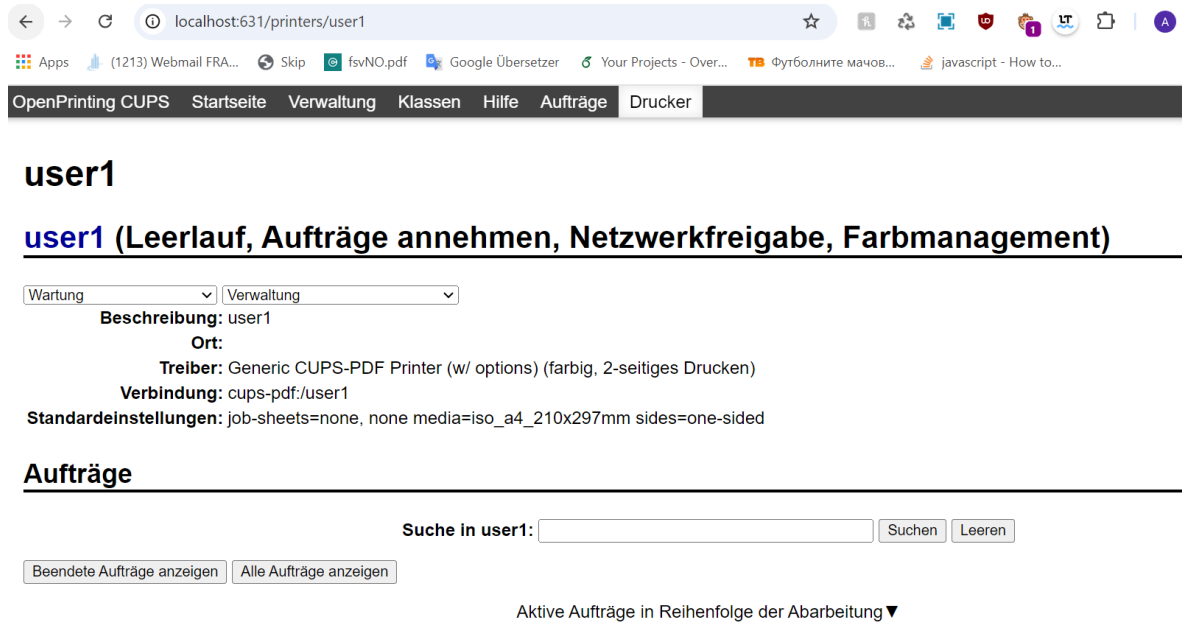


Abbildung 4.8: PDF Drucker für einen Beispielbenutzer

wird der Drucker erstellt, falls er nicht bereits existiert. Als Antwort bekommt der Client die URL des Druckers. Um die VM mit den Drucker zu verbinden, muss der CUPS-PDF Drucker als IPP-Drucker Instanz hinzugefügt werden. Das kann mit einem Befehl im Terminal gemacht werden (Listing 14).

```

1 sudo lpadmin -p <PRINTER_NAME_LOCAL> -E -v
  ↪ ipp://<Container_Adress>:631/printers/<PRINTER_NAME>
  
```

Listing 14: Erstellung von IPP-Drucker Instanz

Als letztes, muss das Problem gelöst werden, wie das fertige PDF im Browser landen kann. Die einfachste Lösung wäre über eine Rest-API nachzufragen ob es für diesen Nutzer neue Dateien gibt. Das ist allerdings nicht besonders elegant. Viel besser wäre es, dass die PDFs sofort nach dem Drucken bei den Nutzer im Browser landen. Um das zu realisieren, wurde ein Postprocessing-Skript verwendet, der immer ausgeführt wird, wenn der CUPS-PDF Drucker einen Auftrag erledigt. Dieser kann in der CUPS-PDF Konfigurationsdatei angegeben werden indem die Zeile `PostProcessing <PFAD-ZUM-SCRIPT>` hinzugefügt wird. Der Postprocessing-Skript bekommt als Parameter unter anderem den Dateipfad. Dieser kann an den Express Server z.B über einer REST-API geschickt werden. Der Server kann den Benutzer aus dem Dateipfad entnehmen und weiß somit, dass es für diesen Benutzer eine neue Datei gibt. Die Datei kann über die WebSocket-Verbindung in einzelne Teile an den Client geschickt werden, was allerdings etwas unsicher ist. Es ist beispielsweise möglich, dass eine sehr große Datei gedruckt wird, die in viele einzelne Teile über die WebSocket Verbindung

an den Client geschickt werden muss. Es kann aber passieren, dass direkt im Anschluss eine kleinere Datei gedruckt wird, die direkt abgeschickt wird, während Teile von der größeren Datei noch geschickt werden. Somit können falsche Teile zu den falschen Dateien hinzugefügt werden, was die Dateien kaputt machen wird. Um dieses Szenario auszuschließen, wird zum Herunterladen von Dateien eine Download-API im Backend benutzt. Mit Express lässt sich leicht eine solche API erstellen, z.B mit `express.static()`. Damit das Backend Zugriff auf den PDFs von den Docker Container hat, kann der Ablageordner als Docker Volume angegeben werden. Somit landen die Dateien auch automatisch auf das Hostsystem, wo das Backend den Zugriff drauf hat. Prinzipiell kann CUPS auch auf dem selben Host wie das Backend installiert werden, was den Kommunikationsprozess erleichtern würde. Da die Komponente aber auch in das DESIGN Projekt integriert wird, ist es jedoch erforderlich sie als einzelne unabhängige Komponente zur Verfügung zu stellen.

## 4.4 Dateitransfer für RDP

RDP ist grundsätzlich in der Lage auch ganze Dateien zu übertragen. Das ist zwar kein Thema dieser Arbeit, da aber die Funktion zum Herunterladen von PDF-Dateien bereits implementiert ist, kann sie auch für andere Dateitypen wiederverwendet werden. Das wäre eine zusätzliche nützliche Funktionalität für den Endbenutzer und benötigt nur wenige Konfigurationen.

Um diese Funktion zu aktivieren, gibt es drei Parameter die gesetzt werden müssen. Der erste ist `enable-drive`, der auf `true` gesetzt werden muss, um die Dateitransfer Option zu aktivieren. Der zweite ist `drive-path`, der den Pfad angibt wo die Dateien auf das Host-System abgelegt werden. Dabei ist zu beachten, dass das Host-System, das System ist, wo `guacd` läuft. In diesem Fall läuft `guacd` im Docker-Container, das heißt die Dateien werden in den Container bleiben. Um das Problem zu umgehen, kann ein Verzeichnis vom Host-System als Docker-Volume angegeben werden. Somit werden die Dateien auch in das Host-System hinterlegt. Der dritte Parameter ist `create-drive-path`. Wenn er auf `true` gesetzt ist, wird der angegebenen Ordner erstellt, falls er noch nicht existiert. Dabei ist zu beachten, dass der Pfad bis zum Ordner existieren muss, da `Guacamole` nur den Endordner erzeugen kann.

Um das Ganze in das Projekt zu integrieren kann folgendes gemacht werden: Auf das Host-System kann einen Ordner erstellt werden, wo die Dateien abgelegt werden können z.B `/home/files`. Beim Container-Erstellung wird er als Volume gemountet z.B mit dem Parameter `-v /home/files:/tmp/files`. Jeder Nutzer soll einen Ordner für seine Dateien haben. Dafür wird aber `guacd` sorgen mithilfe des `create-drive-path` Parameters. Der Parameter `drive-path` muss auf `/tmp/files/<username>` gesetzt werden. Wie der Benutzer auf die Dateien zugreifen kann, gibt es mehrere Möglichkeiten. Die einfachste Lösung wäre eine Fetch- oder Download-API zu implementieren.

Die Benutzer können diese Option verwenden, indem sie mit dem Datei-Manager zu dem `Guacamole`-Laufwerk navigieren (Abbildung 4.9). Dort befindet sich einen Unterordner `Download`. Alle Dateien, die auf dem Laufwerk abgelegt werden (nicht im `Download` Ord-

ner), werden auf dem guacd Host-System (Ordner von den jeweiligen Benutzer) kopiert. Alle Dateien, die im Download-Ordner abgelegt werden, werden direkt an den Web-Client geschickt ohne auf dem Server kopiert zu werden. Der Dateitransfer von Guacd bis zum Webclient verläuft identisch zu dem PDF-Download für RDP. Aus diesem Grund ist keine Implementierung auf der Client-Seite nötig, die `onfile`-Funktion wiederverwendet werden kann. Um den Zugriff auf den Dateien auf dem Server zu ermöglichen, könnten einfache APIs zum Download/Löschen implementiert werden. Da das aber kein Kriterium für das Projekt ist, werden diese APIs nicht implementiert.

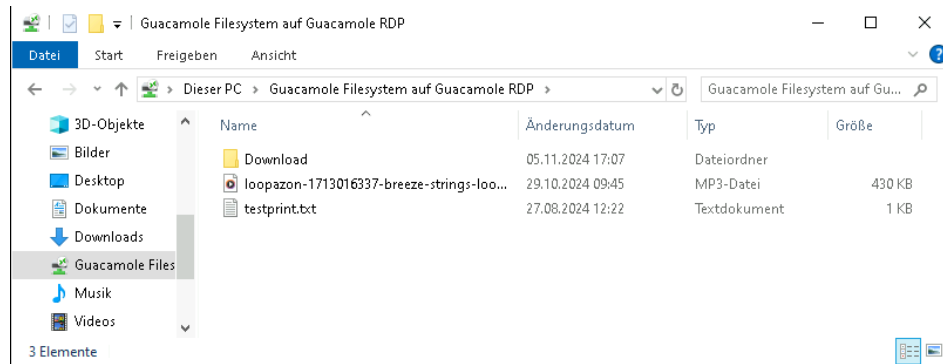


Abbildung 4.9: Guacamole Laufwerk unter RDP

## 4.5 Implementierung der Audioübertragung

Dieser Abschnitt beschäftigt sich mit der Implementierung der Lösung für die Audioübertragung. Dabei gibt es insgesamt vier Fälle, die zu unterscheiden sind: Ein- und-Ausgabe Audio für RDP-Verbindungen und Ein- sowie Ausgabe-Audio für VNC-Verbindungen. Für jede dieser Verbindungsarten müssen spezifische Einstellungen vorgenommen werden, um eine stabile Übertragung zu gewährleisten. Durch diese Unterscheidung kann die Audioübertragung gezielt für jede Verbindung optimiert werden.

Als erstes wird die Audioübertragung für RDP-Verbindungen implementiert. Wie bereits erklärt, unterstützt das RDP-Protokoll die Umleitung von Audio nativ, das heißt dafür müssen keine extra Komponenten entwickelt werden. Guacamole unterstützt ebenfalls die Weiterleitung von den Audiodaten zu dem Web-Client und zurück. Diese Lösung eignet sich sehr gut, da die Audio-Verzögerung sehr klein ist und alles über die selbe Verbindung verläuft und von den RDP-Server verwaltet wird. Somit bleiben Bild und Audio synchron. Das Ausgabe-Audio ist bei RDP-Verbindungen standardmäßig aktiviert. Falls man sie deaktivieren möchte, muss der Verbindungsparameter `disable-audio` auf `false` gesetzt werden. Damit das Audio funktioniert, ist es wichtig den richtigen Audioformat zu benutzen. Den kann man herausfinden indem man in den guacd-Logs bei einer RDP-Verbindung reinschaut (Siehe Listing 15).



```
1 guacd[810]: INFO:      Accepted format: 16-bit PCM with 2 channels at  
   ↪ 44100 Hz
```

Listing 15: Guacd-Log für die Akzeptierten Audioformate

In diesen Fall handelt es sich um das Audioformat Audio/L16 [1]. Dieser muss beim Verbindungsaufbau während der Handshake-Phase als OP-Code an den an den Guacamole-Server geschickt werden. Es ist allerdings wichtig, dass auch die Version von der Frontend-Bibliothek guacamole-common-js den selben Audioformat unterstützt. Listing 16 zeigt wie man die unterstützten Audioformate im Frontend herausfinden kann. In der Version, die für das Projekt verwendet wurde, sind es die Formaten audio/L8 und audio/L16. Um den gewünschten Audioformat an den guacamole-lite Proxy zu senden, kann er einfach an den Token beim Verbindungsaufbau angehängt werden. Mit dieser Änderung, wird das Audio im Browser automatisch abgespielt. Im Frontend muss dafür nichts weiteres gemacht werden.

```
1 // Get supported types: ['audio/L8', 'audio/L16']  
2 const supportedTypes = Guacamole.AudioPlayer.getSupportedTypes();  
3  
4 // Pass the type as parameter  
5 client.connect('token='+this.token+'&GUAC_AUDIO=audio/L16');
```

Listing 16: Verbindungsaufbau mit Angabe des Audioformats

Als nächstes wurde die Mikrofon-Option für RDP-Verbindungen implementiert. RDP unterstützt die Umleitung von den Mikrofon, und Guacamole unterstützt die Weiterleitung. Um das Mikrofon zu aktivieren muss als erstes der Verbindungsparameter `enable-audio-input` auf `true` gesetzt werden. Diese Option ist standardmäßig deaktiviert. Der Rest der Implementierung erfolgt im Frontend. Hierfür gibt es die Klasse `AudioRecorder` aus der `guacamole-common-js` Bibliothek. Für die Aufnahme von Audio wurde die Funktion `recordAudio` (Listing 17) definiert. Sie funktioniert folgendermaßen: Der Client erstellt einen Audio-Stream und übergibt diesen an den `AudioRecorder`. Dieser nimmt das Audio über eine Browser API, die auf das Mikrofon greift, auf und schickt die einzelnen Audio-Chunks an den RDP Server über Guacamole. Wann die Aufnahme beginnt und endet, wird durch den RDP Server gesteuert. Dieser schickt Signalnachrichten, die den Beginn und das Ende der Aufnahme bestimmen. Wenn die Audio Aufnahme beendet ist, werden der Audio-Stream und Recorder geschlossen. Das heißt, bei Bedarf müssen sie neu initialisiert werden. Dafür kann die Callback-Funktion `onclose` der Klasse `AudioRecorder` auf `recordAudio` gesetzt werden. Somit kann bei Bedarf der Mikrofon erneut verwendet werden. Wichtig hier zu beachten ist, dass das Audioformat stimmen muss. Um herauszufinden welche Formate unterstützt sind gibt es die Funktion `AudioRecorder.getSupportedTypes()`. Wichtig ist noch, dass die `recordAudio` Funktion erst dann aufgerufen werden darf, wenn der Client erfolgreich verbunden ist, ansonsten wird die Aufnahme nicht funktionieren. Öffnet man die

Sound-Einstellungen (Abbildung 4.10), sieht man das Windows das Remoteaudio sowohl für Ausgabe als auch für die Eingabe verwendet.

Die bisher beschriebene Implementierung funktioniert für Windows RDP-Verbindungen. Sollte Beispielsweise xRDP für Linux-Verbindungen verwendet werden, werden dafür zusätzliche Konfigurationen benötigt. Um für solche Verbindungen diese Audio-Option zu gewährleisten, wurde ein externer Installationskript [7] verwendet. Damit funktioniert das Audio für Linux-xRDP-Verbindungen genauso wie beim Windows-RDP-Verbindungen.

```
1 //Code ...
2
3 recordAudio = () => {
4     this.audioStreamRDP = this.client.createAudioStream
5     ↪ ("audio/L16;rate=44100,channels=2");
6     this.recorder =
7     ↪ Guacamole.AudioRecorder.getInstance(this.audioStreamRDP,
8     ↪ "audio/L16;rate=44100,channels=2");
9
10    if(!this.recorder)
11        this.audioStreamRDP.sendEnd();
12    else
13        this.recorder.onclose = this.recordAudio;
14 }
15
16 // Code ...
17
18 // Call recordAudio only when the connection is established
19 client.onstatechange = (state => {
20     if(state == 3){
21         this.recordAudio();
22     }
23 })
```

Listing 17: Funktion zur Aufnahme von Audio bei RDP-Verbindungen

Die Implementierung der Audioübertragung für VNC-Verbindungen ist komplizierter, da VNC keine Geräteumleitung unterstützt. Allerdings bietet Guacamole eine Lösung für die Audioausgabe. Das Audio für die meisten Linux Systeme wird von den PulseAudio Service zur Verfügung gestellt. Dieser Service kann das Audio auch über das Netzwerk streamen. Dafür muss sich guacd mit dem Audio Server der jeweiligen Maschine über eine zusätzliche TCP-Verbindung verbinden. So kann er Bild und Audio kombinieren und an den Client weiterleiten. Dafür müssen zuerst TCP-Verbindungen für den PulseAudio Server aktiviert erlaubt werden, indem in der Datei `/etc/pulse/default.pa` die Zeile `load-module-native-protocol-tcp auth-anonymous=1` hinzugefügt werden. Danach muss

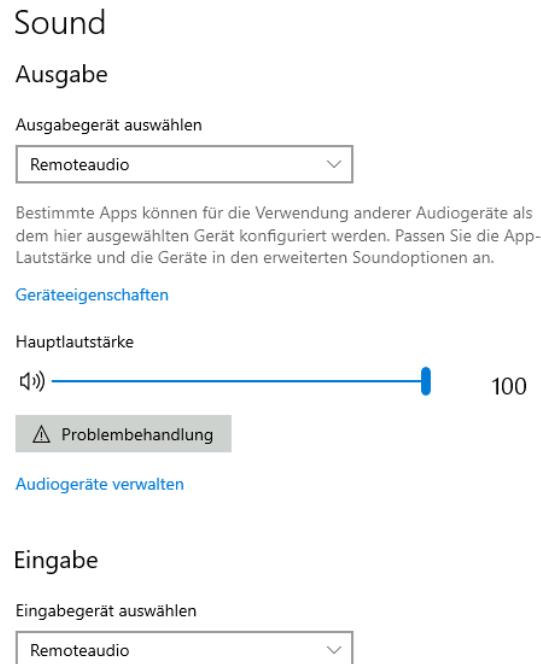


Abbildung 4.10: Remote Audio unter Windows RDP Server

PulseAudio neugestartet werden. Der Server läuft standardmäßig auf Port 4713. Zusätzlich müssen die Verbindungsparametern `enable-audio` auf `true` und `audio-servername` auf die entsprechende Adresse gesetzt werden. Das Audio wird ebenfalls wie beim RDP im Browser automatisch abgespielt.

Für die Benutzung des Mikrofons bietet Guacamole keine ähnliche Lösung. Aus diesem Grund muss eine zweite Verbindung vom Client bis zu der VM aufgebaut werden. Als erstes benötigt aber die VM ein Eingabegerät (Source). Das Problem dabei ist, dass Sources normalerweise Audiodaten aus physische Geräte wie z.B einen Mikrofon erhalten. Das heißt, dass Audiodaten, die aus einem Programm kommen, nicht einfach als Sink-Eingabe benutzt werden können. Es gibt aber einen Weg das zu umgehen, nämlich einen virtuellen Mikrofon zu benutzen. Dafür wurde ein zusätzlicher Sink mit dem Namen `virtual_speaker` erstellt. Das soll nicht als Hauptausgabegerät dienen, also zum Audio abspielen und an `guacd` zu leiten, sondern soll dafür dienen, die Audiodaten vom Browser zu empfangen. Um die empfangenen Daten als Mikrofon-Eingabe zu benutzen wurde das Modul `module-remap-source` benutzt. Damit lässt sich der Monitor (die Ausgabe) von den neuen Sink als Eingabe für einen Mikrofon benutzen. Dieses Mikrofon bekommt den Namen `virtual_mic`. Um die beschriebenen Schritte umzusetzen, wurden in der in der Konfigurationsdatei `/etc/pulse/default.pa` die Zeilen, die in Listing 18 stehen, hinzugefügt. Abbildung 4.11 zeigt, dass die Umleitung der Ausgabe von `virtual_speaker` als Eingabegerät benutzt wird.

```
1 load-module module-null-sink
2 load-module module-null-sink sink_name="virtual_speaker"
  ↳ sink_properties=device.description="virtual_speaker"
3 load-module module-remap-source master="virtual_speaker.monitor"
  ↳ source_name="virtual_mic"
```

Listing 18: Erstellung von virtuellen Mikrofon

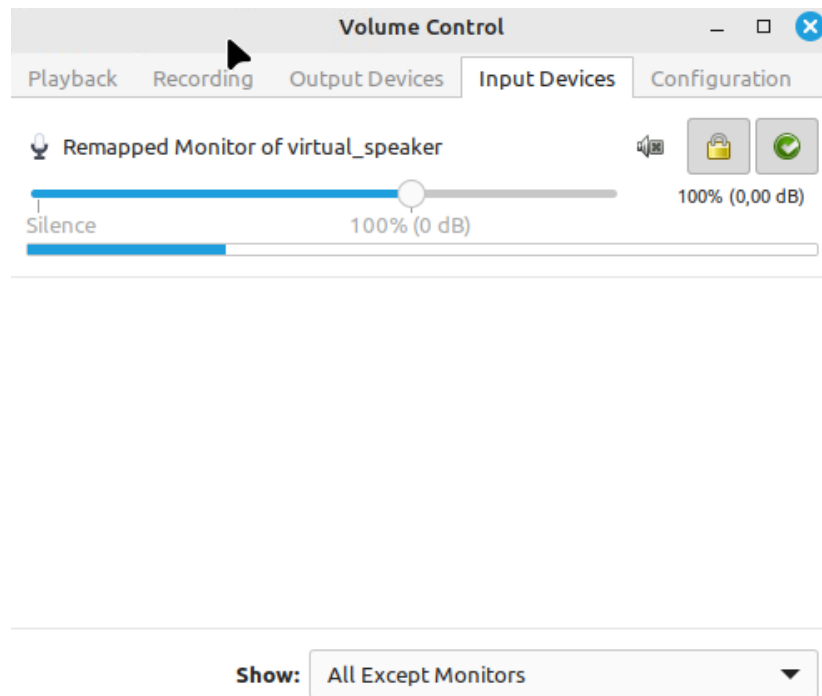


Abbildung 4.11: Erkannte Eingabegeräte von PulseAudio

Als nächstes wurde die Audioaufnahme im Browser implementiert. Dafür wurde einen Client entwickelt, der mit dem Backend über eine WebSocket-Verbindung kommuniziert. Die Audiodaten sollen aus Sicherheitsgründe nicht direkt an der VM geleitet werden. Bei dem Verbindungsaufbau werden einen Authentifizierungstoken und die Adresse des Remote-Desktops übergeben. Der Token wird für diese Anwendung ausgelassen, da keine Authentifizierung implementiert ist. Allerdings verfügt das DESIGN-Projekt über eine Authentifizierung, deswegen wird er als Platzhalter bleiben. Da VNC nicht signalisieren kann wann eine Aufnahme anfängt oder endet, muss das von den Benutzer entschieden werden. Aus diesem Grund wurde einen Button eingebaut, wo beim anklicken die Aufnahme anfängt und endet (Abb. 4.12). Für die Aufnahme selbst steht eine API vom Browser zur Verfügung. Die Implementierung ist in der Klasse `webapp/src/clients/AudioClientNoFilter.js` zu finden. Dabei werden 2048 Bit große Audio-Chunks an das Backend über die WebSocket Verbindung gesendet.

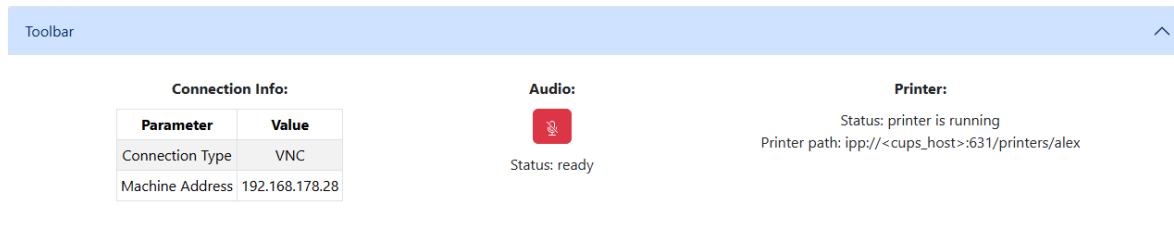


Abbildung 4.12: Toolbar mit Button für Audioaufnahmen

Der letzte Schritt ist, dass die Daten vom Backend an das Mikrofon von der VM zu senden. Dafür gibt es mehrere Möglichkeiten. Eine wäre die Daten über eine TCP Verbindung an einen Server auf der VM zu schicken z.B Netcat. Die angekommenen Daten können dann an den `virtual_speaker` geleitet werden, der als Mikrofon-Eingabe umgeleitet wird (Listing 19). Dabei werden die empfangenen Daten an den Tool `pacat` geleitet, das sie wiederum an den `virtual_speaker` leitet.

```
while true; do nc -l -p "8080" | pacat --format=s16le --channels=1
↪ --rate=44100 --device=virtual_speaker; done
```

Listing 19: Audiodaten Empfang über Netcat

Das ganze funktioniert, allerdings gibt es die Möglichkeit es etwas effizienter zu machen. Es ist nämlich möglich direkt mit PulseAudio über eine TCP Verbindung zu kommunizieren ähnlich wie bei der Audioausgabe. Hierfür kann wieder das Modul `module-protocol-tcp` verwendet werden. Dafür wurde die Zeile `load-module module-native-protocol-tcp auth-anonymous=1 port=8080` in der Konfigurationsdatei von PulseAudio hinzugefügt. Damit kann PulseAudio auf Port 8080 über TCP erreicht werden. Theoretisch kann auch die Server-Instanz auf Port 4713 benutzt werden, wo `guacd` bereits die Audioausgabe entnimmt. Für bessere Performance wird aber die zweite Server Instanz auf Port 8080 benutzt. Jetzt müssen nur noch die Daten vom Backend an den PulseAudio Server geleitet werden. Die direkte Kommunikation zwischen Node und den PulseAudio Server ist aber etwas kompliziert da für eine gute Kommunikation Tools wie `pacat` oder `paplay` benötigt werden, die nur von PulseAudio selbst zur Verfügung gestellt werden. Deswegen wurde folgendes Konzept entwickelt: `pacat` bietet sich gut an um die Audiodaten an der VM zu leiten. Mit dem Parameter `--server` kann der Server auf Port 8080 erreicht werden und mit dem Parameter `--latency-msec` kann die Latenzzeit bestimmt werden. Dabei ist zu beachten, dass wenn die Latenzzeit zu niedrig gesetzt wird, es zu Verschlechterung der Audioqualität kommen kann. Das Backend selbst kann `pacat` als Unterprozess ausführen. Das kann mit den integrierten Modul von Node `child_process` gemacht werden. Voraussetzung dafür ist, dass das unterliegende System über PulseAudio verfügt. Da das Backend als Docker Container ausgeführt wird, ist es unproblematisch PulseAudio auf dem Container mitzuinstallieren. Listing 20 zeigt wie ein `pacat` Unterprozess erstellt werden kann. Dabei werden als Parameter

die Adresse des Servers, Ausgabegerät, Datenformat, Rate, Anzahl der Kanäle und Latenzzeit angegeben. Sobald Daten vom Browser über die Websocket-Verbindung ankommen, werden sie in der Eingabestrom von dem `pacat` Prozess reingeschrieben.

```
1 // Code ...
2
3 const pacatProcess = spawn('pacat', [
4   '--server', destination,
5   '--device', 'virtual_speaker',
6   '--format', 's16le',
7   '--rate', '44100',
8   '--channels', '1',
9   '--latency-msec', '100',
10  ]);
11
12 // Code ...
13
14 ws.on('message', (data) => {
15   if (!pacatProcess.stdin.writable) {
16     console.log('pacat stdin is not writable');
17     return;
18   }
19   pacatProcess.stdin.write(data);
20 });
```

Listing 20: Pacat als Unterprozess für Node

Somit wurde eine Mikrofon-Funktionalität für VNC-Verbindungen eingebaut. Diese läuft komplett unabhängig von Guacamole über eine externe Verbindung. Allerdings entsteht dabei das Problem, dass das Audio etwas verzögert ankommt. Damit diese Verzögerung reduziert wird, wurde der Lanczos-Interpolation-Algorithmus [62] benutzt. Dabei handelt es sich um ein Algorithmus aus der Signalverarbeitung, der den Lanczos Kernel verwendet um Werte zwischen vorhandenen Audioproben durch glatte Interpolation zu berechnen. Der Lanczos-Kernel basiert auf eine modifizierten Sinc-Funktion, die hilft, Verzerrungen und Aliasing-Artefakte zu minimieren, was die Qualität und Effizienz der Audioübertragung verbessert. Die Klasse `AudioRecorder` von `guacamole-common-js` nutzt bereits diesen Algorithmus für Audioaufnahmen, die über die Guacamole-Verbindung gesendet werden. Da der `AudioRecorder` nicht für die externe Verbindungen benutzt werden kann, wurde die Implementierung des Algorithmus aus der Klasse übernommen und für den VNC Audio Client benutzt (Klasse `webapp/src/clients/AudioClientLanczos.js`). Falls man den Lanczos-Interpolation-Algorithmus nicht benutzen möchte, kann die Klasse `AudioClientNoFilter.js` im selben Ordner benutzt werden.

## 4.6 Sicherheit

Sicherheit ist grundsätzlich kein Hauptkriterium für diese Arbeit. Allerdings sind minimale Maßnahmen dennoch wichtig, da sensible Daten wie z.B. Passwörter übertragen werden. Außerdem werden einige Funktionalitäten vom Browser geblockt, wenn die Verbindung unsicher ist, wie z.B. das Herunterladen von Dateien und den Zugriff auf das Mikrofon. Bis jetzt läuft alles über unverschlüsselte Verbindungen. Für die Verbindungen vom Backend zu dem CUPS-Server oder zu den VMs ist das unproblematisch, da das Ganze nicht nach außen geöffnet ist. Die Verbindung vom Frontend zu Backend sollte aber verschlüsselt sein. Aus diesem Grund wurden diese Verbindungen von HTTP zu HTTPS bzw. von WS zu WSS geändert. Dazu wurde ein SSL/TLS-Zertifikat mit OpenSSL [41] nach Anleitung [58] erstellt und konfiguriert. Dafür wird ein Privater Schlüssel für den Server erstellt. Anschließend wird eine Zertifikatsanforderung (CSR) erstellt, die im dritten Schritt unter Verwendung des privaten Schlüssels signiert wird, um das endgültige Zertifikat zu erstellen.

```
1 #STEP 1: Create the server private key
2 openssl genrsa -out cert.key 2048
3
4 #STEP 2: Create the certificate signing request (CSR)
5 openssl req -new -key cert.key -out cert.csr
6
7 #STEP 3: Sign the certificate using the private key and CSR
8 openssl x509 -req -days 3650 -in cert.csr -signkey cert.key -out
   ↪ cert.crt
```

Listing 21: Erzeugung von TLS/SSL Zertifikat mit OpenSSL

Dabei handelt es sich um ein selbst signiertes Zertifikat, deswegen warnt der Browser, dass die Webseite nicht sicher ist. Für Entwicklungszwecke wird ein solches Zertifikat ausreichen. Für die Produktion sollte ein von einer offiziellen Zertifizierungsstelle erstelltes Zertifikat benutzt werden. Mit den SSL-Optionen kann der Express HTTP-Server zu HTTPS geändert werden. Im Frontend müssen die URLs entsprechend angepasst werden. Abbildung 4.13 zeigt den Zertifikat, der im Browser abgerufen werden kann. Der Zertifikat enthält Informationen über den Eigentümer des Servers, Aussteller des Zertifikats, die Gültigkeitsdauer und den öffentlichen Schlüssel des Servers.

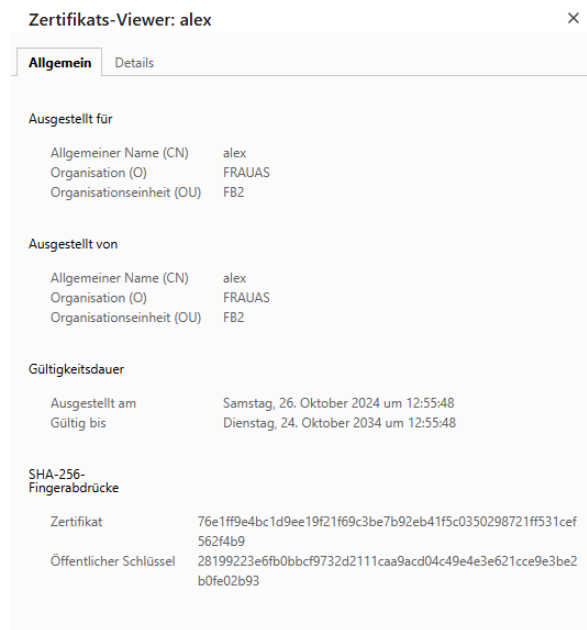


Abbildung 4.13: TLS Zertifikat von den Server

## 4.7 Bereitstellung

Als Letztes wurde die Implementierte Anwendung bereitgestellt. Dafür werden die Komponenten als Docker Container zur Verfügung gestellt. Somit können es Anwender leicht starten und damit arbeiten ohne irgendetwas zusätzlich zu installieren. Alles was benötigt wird, soll in den Docker-Containern vorhanden sein. Folgende Komponenten sollen zur Verfügung gestellt werden: Frontend und Backend der Web-Anwendung, CUPS-Server und Guacd. Die Remote-Desktops werden von den Benutzern selbst vorbereitet, das heißt installation von VMs, Aufsetzung von RDP/VNC Sever sowie PulseAudio Einstellungen und Hinzufügen von IPP-Drucker für VNC.

Als erstes wird die Web-Anwendung bereitgestellt. Das Front und Backend können als einzelne Container laufen. Effektiver wäre es beide als eine Anwendung auszuführen. Um das React-Frontend in das Express-Backend zu integrieren, wurde folgendes gemacht: Zuerst wurde die React-Anwendung mithilfe des Befehls `npm run build` gebaut. Dadurch wurde ein `build` Ordner erstellt, der alle statischen Dateien enthält, die für die Ausführung der Anwendung im Browser erforderlich sind. Statische Dateien sind Dateien, die ohne dynamische Bearbeitung direkt vom Server an den Browser gesendet werden, wie z. B. HTML-Dateien oder gebündelte JavaScript-Dateien. Als nächstes wurde der Inhalt des `build`-Ordners in den Unterordner `public` vom Backend kopiert. Zum Schluss wurden die Dateien mithilfe von `app.use(express.static('./public'))`; bereitgestellt. Der Express Server stellt somit nicht nur die API Endpunkte bereit, sondern liefert auch die statischen Dateien der React-App. Damit der Benutzer Node.js nicht lokal installieren muss um die Anwendung zu bauen, wurde alles in den Dockerfile definiert. Zum Schluss erhält man einen Container, der



die ganze Webanwendung enthält.

Der CUPS-Server steht bereits als Docker-Container zur Verfügung. Im Prinzip kann er auch in den selben Container laufen. Da aber die Komponente auch für das DESIGN Projekt benutzt wird, soll sie unabhängig von den Rest laufen und wird in einen eigenen Container ausgeführt. Damit das Backend auf die PDF Dateien Zugriff hat, müssen beide Container einen gemeinsamen Docker-Volume verwenden. Für guacd kann das offizielle Docker-Image von Apache Guacamole benutzt werden. Eine genaue Anleitung wie das Projekt reproduziert werden kann, ist in den Projektordner als README-Datei vorhanden. Abbildung 4.14 fasst die Container-Architektur zusammen.

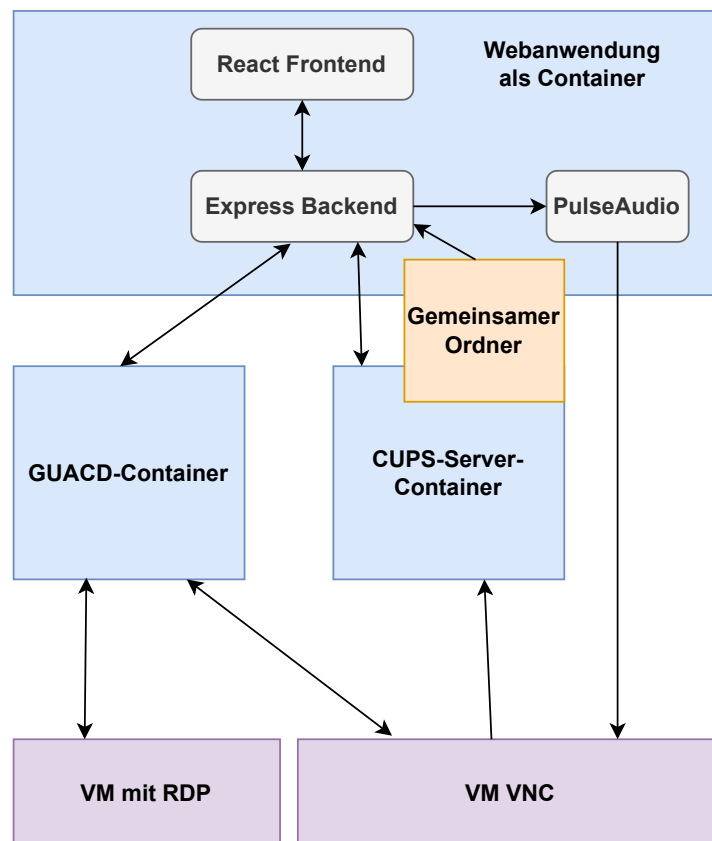


Abbildung 4.14: Container-basierte Ansicht der Anwendung

# Kapitel 5

## Bewertung

Dieses Kapitel beschäftigt sich mit den Ergebnissen dieser Arbeit. Zunächst wird erläutert, welche Ziele erreicht wurden und welche nicht. Danach werden die entwickelten Methoden getestet und ausgewertet. Zum Schluss wird deren Bedeutung für die Forschung im Bereich DaaS diskutiert.

### 5.1 Ziele und Ergebnisse

Das Ziel dieser Arbeit war, eine Webanwendung zu implementieren, über die einen Desktop ferngesteuert werden kann. Die Webanwendung sollte folgende Funktionalitäten bereitstellen: Bildübertragung, Mausinteraktionen, Tastatureingaben, Audioübertragung in beide Richtungen sowie eine Druckoption. Diese Funktionalitäten sollten für VNC- und RDP-Verbindungen vorhanden sein. Außerdem sollten die einzelnen Module wiederverwendbar sein, da sie in das DESIGN Projekt integriert werden müssen.

Die Basis-Anwendung wurde erfolgreich implementiert und alle Kriterien dafür wurden erfüllt. Ein Benutzer kann sich mit Desktops verbinden und sie steuern. Alles was er dafür benötigt sind die Daten von den Remote-Gerät, wie IP-Adresse, Benutzername und Passwort. Daraus kann ein verschlüsseltes Token generiert werden, mit dem der Remote-Desktop erreicht werden kann. Zusätzlich wurde die Verbindung zwischen dem Webbrowser und dem Backend mit TLS verschlüsselt, womit auch die Sicherheitskriterien erfüllt wurden.

Die Druck-Funktionalität wurde ebenfalls erfolgreich implementiert. Für RDP-Verbindungen wurde eine Option implementiert, mit der Druckaufträge über guacd als PDF-Dateien an den Web-Client gesendet werden können. So kann der Benutzer seine ausgewählten Dateien an den Guacamole-Drucker senden, und sie dann sofort als PDF im Browser herunterladen. Für VNC-Verbindungen wurde ein anderes Konzept verwendet, da es dafür keine Druckerumleitung gibt. Es wurde ein CUPS-Server aufgesetzt, der für jeden Nutzer einen PDF-Drucker verwaltet. Der Endnutzer muss prinzipiell nur einen Namen für den Drucker angeben und ihn als IPP-Drucker auf dem Remote-Desktop hinzufügen. So kann er Aufträge an seinen Drucker schicken und sie dann im Browser herunterladen. Das Erstellungsprozess von Dru-

ckern wurde mit einem Shell-Skript automatisiert, damit sie nicht manuell erstellt werden müssen. Die Druck-Optionen für RDP- und VNC-Verbindungen erfolgreich implementiert und in das DESIGN Projekt integriert.

Die Erweiterung um Ein- und Ausgabe-Audio war ebenfalls erfolgreich. Für RDP-Verbindungen wurde das RDP-Audio verwendet, das von `guacd` an den Browser weitergeleitet wird. Die Audioausgabe von dem Remote-Desktop wird im Browser abgespielt und beim Audioaufnahmen greift der Browser auf das lokale Mikrofon und sendet die Daten an dem Desktop. Für VNC-Verbindungen wurde ebenfalls eine Lösung für die Audioübertragung erfolgreich entwickelt. Dafür wurde der PulseAudio-Server benutzt. Für die Audioausgabe, verbindet sich `guacd` mit dem PulseAudio-Server und leitet die empfangenen Audio-Daten von den Remote-Gerät an den Browser weiter, wo sie abgespielt werden. Für Audioaufnahmen, muss der Benutzer den Mikrofon im Browser selbst über einen Button starten. Die aufgenommenen Daten werden über eine zusätzliche Verbindung an den Remote-Desktop geleitet. Für niedrigere Latenz wurde das Lanczos-Resampling angewendet. Auch die Audio-Lösung wurde auch erfolgreich in das DESIGN-Projekt integriert.

Zusätzlich wurde eine Datei-Transfer Option für RDP-Verbindungen eingebaut. Der Benutzer kann beliebige Dateien in das Guacamole-Laufwerk hochladen und sie dann im Browser herunterladen. Das ist zwar kein Kriterium für das Projekt, bietet jedoch eine sehr nützliche Funktion für die Benutzer. Die Anwendung ist auf Github verfügbar und kann einfach als Docker-Container gestartet werden.

## 5.2 Bewertung der Methodik

In diesem Abschnitt werden die ausgewählten Methoden bewertet. Bezüglich der Basis-Anwendung standen nicht viele Optionen zur Auswahl, da die Voraussetzung war, dass sie auf Guacamole basiert. Zur Auswahl stand nur welche Frameworks und Bibliotheken verwendet werden. Es wurde dazu entschieden die `guacamole-lite`-Bibliothek für das Backend zu benutzen. Der `guacamole-lite`-Server ist leicht zu benutzen und lässt sich einfach in `Express.js` Anwendungen integrieren. Insgesamt erwies sich diese Lösung als ausreichend, um die Anforderungen zu erfüllen. Ein möglicher Nachteil gegenüber der offiziellen `guacamole-common`-Bibliothek besteht darin, dass `guacamole-lite` nicht Teil des offiziellen Guacamole-Projekts ist und daher eventuell eines Tages nicht mehr weiterentwickelt wird.

Um die Druck-Funktionalität zu testen, wurden Testdateien mit beiden beschriebenen Konzepten gedruckt. Es wurden dafür eine kleine Datei (Textdatei, 1 Seite, 46 Byte) und eine große Datei (Buch als PDF, 196 Seiten, 5 MB) vorbereitet. Die Zeitmessung begann jeweils ab dem Zeitpunkt, an dem der Benutzer auf Drucken klickte, und endete, sobald das Download-Dialogfenster im Browser erschien. Beim Drucken der kleine Datei brauchten beide Druck-Konzepte nicht besonders lange. Beim Drucken über CUPS dauerte es 3,3 Sekunden und die heruntergeladene Datei hatte eine Größe von 57,6 KB. Beim RDP-Drucken dauerte es 6,0 Sekunden und die heruntergeladene Datei hatte eine Größe von 7,65 KB. Beim Drucken des Buchs dauerte es deutlich länger. Die CUPS-Methode brauchte 3:32,00 Minuten und die heruntergeladene Datei war 42 MB groß. Beim RDP-Drucken dauerte es

4:05,00 Minuten und die Datei war 50 MB groß. Im Gegensatz zu Drucken geht es mit dem RDP-Datei Transfer deutlich schneller. Die Übertragung des Buchs als Datei, dauerte 16,00 Sekunden und die Datei behielt ihre Größe. Abschließend ist zu beachten, dass die angegebenen Zeiten je nach Hardware und Netzwerkverbindung variieren können.

Für das Testen der Audiowiedergabe wurden verschiedene Töne und Videos auf den VMs abgespielt. Bei beiden Verbindungsarten konnte keine spürbare Latenz festgestellt werden und die Töne waren synchron zum Bild. Beim Audioaufnehmen zeigte das RDP-Audio ein insgesamt gutes Ergebnis. Es gingen keine Audiodaten verloren und die Latenz war gering. Der Hauptvorteil dabei ist die Signalisierung, wann die Aufnahmen beginnen und enden. Somit muss der Benutzer den Mikrofon nicht manuell an- und ausschalten. Beim VNC-Verbindungen gibt es solche Signalisierungen nicht, weshalb der Benutzer das Mikrofon im Browser manuell aktivieren muss. Die Latenz bleibt auch hier gering, jedoch fehlt ein Mechanismus, der sicherstellt, dass Bild und Audioaufnahme synchron bleiben.

Insgesamt lässt sich sagen, dass RDP-Verbindungen etwas besser für diesen Anwendungsfall geeignet sind. Zum einen bietet RDP mehr Funktionalitäten als VNC und die Datenübertragung verläuft über eine einzige Verbindung. Das macht die Gesamtarchitektur der Anwendung einfach und sicher. Zum anderen ist die Audioaufnahme-Funktionalität aufgrund des Signalisierungsmechanismus für Aufnahmebeginn und -ende besser. Durch das Installationsskript für xRDP können auch Linux-Desktops die Funktionalitäten benutzen, die der Windows-RDP Server bietet.

### 5.3 Bedeutung der Ergebnisse

Mit den Ergebnissen dieser Arbeit wurde gezeigt, dass webbasierte Remote-Desktop-Viewer um zusätzliche Funktionalitäten erweitert werden können, die entscheidend für eine positive Benutzererfahrung sind. Die Erweiterung um Audio ermöglicht die effiziente Nutzung von Anwendungen, die besonders gut für Remote-Desktop-Umgebungen geeignet sind, wie beispielsweise Audio- oder Videobearbeitungssoftware. Solche Anwendungen benötigen eine hohe Rechenleistung, die nicht alle Geräte bereitstellen können. Daher profitieren Benutzer besonders von Lösungen, die in der Lage sind, Audio zu leiten. Die eingebaute Druckoption ist ebenfalls wichtig für eine gute Benutzererfahrung. Ohne sie müssten Benutzer sich die Dateien z.B über E-Mail senden um sie lokal auszudrucken. Das nimmt Zeit in Anspruch und ist ineffizient. Die eingebaute Druck-Funktionalität ist besonders nützlich für Anwendungen, die Dokumente erzeugen, die auf Papier ausgedruckt werden müssen. Es wurde außerdem wurde gezeigt, dass das Guacamole sich effizient in selbstgeschriebenen Webanwendungen integrieren lässt. Das bedeutet, dass Firmen deren Mitreitern Remote-Desktop-Lösungen anbieten können, die in deren eigenen Webanwendungen integriert sind. Sie können deren eigene Authentifizierung benutzen und müssen sich nicht um die Verwaltung von zusätzlichen Anwendungen kümmern.

# Kapitel 6

## Schluss

### 6.1 Zusammenfassung

Diese Arbeit beschäftigte sich mit der Entwicklung von einen webbasierten Remote-Desktop-Viewer und deren Erweiterung um Audio und Drucken. Als Voraussetzung galt den Guacamole Server (guacd) als Vermittler zu benutzen. In der Literatur wurden einige Arbeiten gefunden, die Lösungen basierend auf das Guacamole-Protokoll vorstellen. Diese beschäftigen sich aber hauptsächlich mit der Bereitstellung der offiziellen Guacamole-Webanwendung. Zudem wurde nach mögliche Lösungen für die Erweiterung um Audio und Drucken untersucht. Für die Audioübertragung wurden Ansätze wie RDP-Audio, WebRTC, SPICE-Protokoll und PulseAudio-Server gefunden. Für Drucken kamen Konzepte wie Cloud-Printing, das IPP-Protokoll, RDP-Druckerumleitung oder der Einsatz von CUPS-Server in Betracht.

Für die Umsetzung des Projekts, wurde ein Proxmox-VE Server aufgesetzt, wo zwei VMs (eine mit Windows und eine mit Linux) laufen. Sie wurden mit RDP und VNC Server ausgestattet. Für die Fernsteuerung von den VMs, wurde eine Webanwendung mit Express.js und React geschrieben. Als Proxy-Komponente zwischen guacd und das Frontend wurde einen guacamole-lite Server benutzt.

Für die Implementierung von Audioübertragung, wurde bei RDP-Verbindungen das RDP-Audio sowohl für die Ausgabe als auch für Aufnahmen verwendet. Für VNC-Verbindungen wurde der PulseAudio Server angewendet. Dabei wurden die Audioausgabe-Daten an guacd geleitet, der sie mit den VNC-Daten kombiniert und an den Client sendet. Bei der Mikrofon-Benutzung wurden die Audiodaten vom Browser über eine zusätzliche WebSocket-Verbindung an das Backend gesendet. Von dort aus wurden sie mithilfe eines pacat-Unterprozess an den PulseAudio-Server der VM geschickt.

Bezüglich des Druckens, wurde bei RDP-Verbindungen die eingebaute Druckerumleitung-Funktion benutzt. Dabei werden die Druckaufträge mithilfe von GhostScript zu PDF umgewandelt und an den Client geschickt. Für VNC-Verbindungen wurde ein CUPS-Server bereitgestellt. Darauf laufen virtuelle Drucker (CUPS-PDF), die ebenfalls die Aufträge zu PDF umwandeln und an den Clients versendet. Zusätzlich wurden alle Verbindungen vom

Backend zum Frontend mit HTTPS gesichert. Die gesamte Anwendung wurde als Docker-Containern zur Verfügung gestellt. Die Ergebnisse dieser Arbeit zeigten, dass für webbasierte Remote-Desktop-Lösungen viele wichtige zusätzliche Funktionalitäten eingebaut werden können.

## 6.2 Ausblick

DaaS ist ein aktuelles Thema, das weiterhin großes Potenzial für Entwicklungen und Erweiterungen bietet. Die Anwendung wurde so implementiert, dass sie für zukünftige Erweiterungen offen ist. In dieser Arbeit wurden Lösungen für Audioübertragung und Drucken entwickelt, ein Desktop verfügt aber um viele weitere Funktionalitäten, die wichtig für die Benutzererfahrung sind.

Eine mögliche Erweiterung für zukünftige Arbeiten wäre, eine bessere Lösung für Audioaufnahmen bei VNC-Verbindungen zu entwickeln. Die Lösung von dieser Arbeit funktioniert und erfüllt die Anforderungen, allerdings ist kein Mechanismus vorhanden, der signalisiert wann ein Benutzer auf das Mikrofon greift. So muss er den Mikrofon immer manuell an und ausschalten. Ein automatisiertes An- und Ausschalten von den Mikrofon wie beim RDP würde die Benutzererfahrung verbessern und die Bedienung der Anwendung erleichtern. Eine weitere mögliche Erweiterung wäre, eine Funktionalität für Datei-Transfer bei VNC-Verbindungen zu entwickeln. Für RDP-Verbindungen ist diese Option bereits integriert. Für VNC bietet Guacamole die Möglichkeit, Dateien über das Secure File Transfer Protocol (SFTP) zu übertragen [21]. Eine Erweiterung um einer Zwischenablage (Clipboard) könnte die Benutzerfreundlichkeit ebenfalls deutlich verbessern. Damit könnten Nutzer Inhalte wie Texte oder Bilder schnell und einfach zwischen ihrem Host-System und dem Remote-Desktop kopieren und einfügen. Guacamole bietet dafür bereits Lösungen an, die sich in der Anwendung integrieren lassen [21]. Die Nutzung von der lokalen Webkamera für Videoaufnahmen könnte ebenfalls eine mögliche Erweiterung für die Anwendung sein. Das wäre besonders nützlich bei Videoverarbeitung auf dem Remote-Desktop. Eine bessere Anpassung der Bildschirmgröße wäre ebenfalls denkbar. Auch Sicherheit ist ein Thema, wo noch viel entwickelt werden kann. Eine Registrierung und Anmeldung der Nutzer wären wichtige Punkte, die in der aktuellen Anwendung fehlen. Auch eine Absicherung der Endpunkte im Backend z.B mit Tokens würde die Sicherheit erheblich verbessern. Auch Dienste wie der CUPS-Server und PulseAudio sollten geschützt werden, um potenzielle Angriffsflächen zu minimieren. Insgesamt gibt es viele Ideen und Möglichkeiten, die Anwendung in Zukunft weiterzuentwickeln und ihre Funktionen zu erweitern.

## 6.3 Fazit

In dieser Arbeit wurde gezeigt, wie mithilfe des Guacamole-Protokolls Desktops über den Internetbrowser ferngesteuert werden können. Dies ermöglicht eine benutzerfreundliche und plattformübergreifende Lösung für den Zugriff auf Remote-Desktops, ohne dass

zusätzliche Software auf dem Endgerät installiert werden muss. Darüber hinaus wurde gezeigt, wie zusätzliche Funktionen wie Audio- und Druckunterstützung sowie Dateitransfer integriert werden können, um die Benutzererfahrung zu verbessern und die Einsatzgebiete von Remote-Desktop-Lösungen zu erweitern. Die Ergebnisse dieser Arbeit zeigen, dass browserbasierte Remote-Desktop-Viewer ein großes Potential haben und sich als zukunftsfähige Lösungen anbieten. Zudem wurde verdeutlicht, dass solche Lösungen sich leicht in eigene Webanwendungen integrieren lassen. So können Firmen deren Mitarbeitern kostengünstige effiziente Lösungen für einfaches Remote-Arbeiten anbieten. Zukünftige Arbeiten könnten sich auf die Optimierung und Erweiterung der bestehenden Funktionen konzentrieren, beispielsweise durch eine verbesserte Mikrofon-Integration für VNC-Verbindungen oder die Implementierung weiterer Features wie Webcam-Unterstützung, Dateiübertragung für VNC-Verbindungen, oder Clipboard-Funktionalität. Zudem könnten auch Sicherheitsaspekte weiter ausgebaut werden, um die Datenintegrität und den Datenschutz noch stärker zu gewährleisten.

# Anhang A

## Programmcode

Das gesamte Projekt steht auf GitHub zur Verfügung und ist unter [3] zugänglich.



# Literatur

- [1] Harald T. Alvestrand und James Salsman. *The Audio/L16 MIME content type*. RFC 2586. Mai 1999. DOI: 10.17487/RFC2586. URL: <https://www.rfc-editor.org/info/rfc2586>.
- [2] AnyDesk. *Remote Desktop Web Client*. 2024. URL: <https://anydesk.com/en/goanydesk> (besucht am 30.10.2024).
- [3] Alexander Atanassov. *Desktop As A Service*. URL: <https://github.com/alllexander1/DesktopAsAService> (besucht am 05.11.2024).
- [4] Christian Baun und Johannes Bouché. "Closing the Gap between Web Applications and Desktop Applications by Designing a Novel Desktop-as-a-Service (DaaS) with Seamless Support for Desktop Applications". In: *Open Journal of Cloud Computing (OJCC)* 8.1 (2023), S. 1–19.
- [5] Carl Boettiger. "An introduction to Docker for reproducible research". In: *ACM SIGOPS Operating Systems Review* 49.1 (2015), S. 71–79.
- [6] Peter Buxmann, Thomas Hess und Sonja Lehmann. "Software as a Service". In: *Wirtschaftsinformatik* 50 (2008), S. 500–503.
- [7] c-nergy. *xRDP – Easy install xRDP on Ubuntu*. 2024. URL: <https://c-nergy.be/blog/?p=19814> (besucht am 26.07.2024).
- [8] Matthias Kevin Caspers. "React and redux". In: *Rich Internet Applications w/HTML and Javascript* 11 (2017).
- [9] Simon MC Cheng. *Proxmox High Availability*. Packt Publishing Ltd, 2014.
- [10] Cloudflare. *Was ist TLS (Transport Layer Security)?* 2024. URL: <https://www.cloudflare.com/de-de/learning/ssl/transport-layer-security-tls/> (besucht am 27.10.2024).
- [11] Cloudflare. *Was passiert bei einem TLS-Handshake? — SSL-Handshake*. 2024. URL: <https://www.cloudflare.com/de-de/learning/ssl/what-happens-in-a-tls-handshake/> (besucht am 27.10.2024).
- [12] Developer.Mozilla. *Express/Node introduction*. 2024. URL: [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction) (besucht am 07.10.2024).
- [13] *Directprint*. 2024. URL: <https://directprint.io/> (besucht am 30.10.2024).

- [14] Docker. *Docker Docs*. 2024. URL: <https://docs.docker.com/> (besucht am 12.10.2024).
- [15] Docker. *Docker Desktop WSL 2 backend on Windows*. URL: <https://docs.docker.com/desktop/wsl/> (besucht am 01.11.2024).
- [16] Dockerhub. *guacamole/guacd*. 2024. URL: <https://hub.docker.com/r/guacamole/guacd> (besucht am 06.08.2024).
- [17] Ezeep. *Remote-Desktop-Drucken*. 2024. URL: <https://www.ezeep.com/de/loesung/remote-desktop-drucken/> (besucht am 30.10.2024).
- [18] Facebook. *React*. 2024. URL: <https://github.com/facebook/react> (besucht am 07.10.2024).
- [19] How To Forge. *So installierst du Apache Guacamole als Docker Container unter Ubuntu*. 2022. URL: <https://www.howtoforge.de/anleitung/so-installierst-du-apache-guacamole-als-docker-container-unter-ubuntu/> (besucht am 13.10.2024).
- [20] Apache Software Foundation. *Apache Guacamole*. 2024. URL: <https://guacamole.apache.org/#:~:text=Apache%20Guacamole%20is%20a%20clientless,desktops%20is%20a%20web%20browser.> (besucht am 30.10.2024).
- [21] The Apache Software Foundation. *Configuring Guacamole*. 2024. URL: <https://guacamole.apache.org/doc/gug/configuring-guacamole.html#vnc> (besucht am 04.11.2024).
- [22] The Apache Software Foundation. *guacamole-common*. 2024. URL: <https://guacamole.apache.org/doc/gug/guacamole-common.html> (besucht am 19.08.2024).
- [23] The Apache Software Foundation. *guacamole-common-js*. 2024. URL: <https://guacamole.apache.org/doc/gug/guacamole-common-js.html> (besucht am 19.08.2024).
- [24] The Apache Software Foundation. *Guacamole: Implementation and architecture*. URL: <https://guacamole.apache.org/doc/gug/guacamole-architecture.html> (besucht am 31.10.2024).
- [25] The Apache Software Foundation. *The Guacamole Protocol*. URL: <https://guacamole.apache.org/doc/gug/guacamole-protocol.html> (besucht am 31.10.2024).
- [26] The Apache Software Foundation. *What is Guacamole*. URL: <https://guacamole.apache.org/doc/gug/introduction.html> (besucht am 31.10.2024).
- [27] Sebastián García u. a. "Remote lab access: a powerful tool beyond the pandemic". In: *2022 Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (XV Technologies Applied to Electronics Teaching Conference)*. IEEE. 2022, S. 1–5.
- [28] Victor Gaydov. *PulseAudio under the hood*. 2017. URL: <https://gavv.net/articles/pulseaudio-under-the-hood/> (besucht am 29.10.2024).

- [29] Google. *Chrome Remote Desktop*. URL: <https://remotedesktop.google.com/?hl=de> (besucht am 30.10.2024).
- [30] Cloud Guru. *Working with the CUPS Print Server*. URL: <https://www.pluralsight.com/cloud-guru/labs/aws/working-with-the-cups-print-server> (besucht am 05.11.2024).
- [31] Ismail Hassan. "Leveraging Apache Guacamole, Linux LXD and Docker Containers to Deliver a Secure Online Lab for a Large Cybersecurity Course". In: *2022 IEEE Frontiers in Education Conference (FIE)*. IEEE. 2022, S. 1–9.
- [32] IBM. *Was ist Desktop-as-a-Service (DaaS)?* 2024. URL: <https://www.ibm.com/de-de/topics/desktop-as-a-service> (besucht am 29.10.2024).
- [33] *Implement Spice protocol support*. 2022. URL: <https://github.com/apache/guacamole-server/pull/394> (besucht am 31.10.2024).
- [34] Marián Jančík, Lucia Melníková und Doc Ing Branislav Kandra. "Transformation of eurocontrol's classroom courses into remote learning". In: *2021 New Trends in Aviation Development (NTAD)*. IEEE. 2021, S. 66–69.
- [35] Kasm. *Kasm Workspaces Documentaion*. 2023. URL: <https://www.kasmweb.com/docs/latest/index.html> (besucht am 30.10.2024).
- [36] Alexey Melnikov und Ian Fette. *The WebSocket Protocol*. RFC 6455. Dez. 2011. DOI: 10.17487/RFC6455. URL: <https://www.rfc-editor.org/info/rfc6455>.
- [37] Microsoft. *Remotedesktopprotokoll*. 2023. URL: <https://learn.microsoft.com/de-de/windows/win32/termserv/remote-desktop-protocol?redirectedfrom=MSDN> (besucht am 16.07.2024).
- [38] Microsoft. *Microsoft Universal Print*. 2024. URL: <https://learn.microsoft.com/de-de/universal-print/discover-universal-print> (besucht am 30.10.2024).
- [39] Davide Mulfari, Antonio Celesti und Massimo Villari. "A computer system architecture providing a user-friendly man machine interface for accessing assistive technology in cloud computing". In: *Journal of Systems and Software* 100 (2015), S. 129–138.
- [40] Davide Mulfari u. a. "Using virtualization and guacamole/vnc to provide adaptive user interfaces to disabled people in cloud computing". In: *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*. IEEE. 2013, S. 72–79.
- [41] OpenSSL. *openssl*. URL: <https://github.com/openssl/openssl> (besucht am 03.11.2024).
- [42] Linda Dailey Paulson. "Building rich web applications with Ajax". In: *Computer* 38.10 (2005), S. 14–17.
- [43] Christian Peters. "Building rich internet applications with node.js and express.js". In: *Rich Internet Applications w/HTML and Javascript* (2017), S. 15.
- [44] Vadim Pronin. *guacamole-lite*. URL: <https://github.com/vadimpronin/guacamole-lite> (besucht am 03.11.2024).

- [45] Proxmox. *Qemu/KVM Virtual Machines*. 2022. URL: [https://pve.proxmox.com/wiki/Qemu/KVM\\_Virtual\\_Machines](https://pve.proxmox.com/wiki/Qemu/KVM_Virtual_Machines) (besucht am 16.07.2024).
- [46] Proxmox. *Proxmox Virtual Environment*. 2024. URL: <https://www.proxmox.com/de/proxmox-virtual-environment/uebersicht> (besucht am 16.07.2024).
- [47] PyPI. *pyguacamole*. 2021. URL: <https://pypi.org/project/pyguacamole/> (besucht am 19.08.2024).
- [48] Eric Rescorla. *HTTP Over TLS*. RFC 2818. Mai 2000. DOI: 10.17487/RFC2818. URL: <https://www.rfc-editor.org/info/rfc2818>.
- [49] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. URL: <https://www.rfc-editor.org/info/rfc8446>.
- [50] Tristan Richardson u. a. "Virtual network computing". In: *IEEE Internet Computing* 2.1 (1998), S. 33–38.
- [51] Peter Saint-Andre u. a. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. RFC 6202. Apr. 2011. DOI: 10.17487/RFC6202. URL: <https://www.rfc-editor.org/info/rfc6202>.
- [52] Cybele Software. *ThinVNC HTML5 Remote Access*. 2014. URL: [https://files.cybelesoft.com/docs/thinvnc\\_guide.pdf](https://files.cybelesoft.com/docs/thinvnc_guide.pdf) (besucht am 10/2024).
- [53] SPICE. *Spice User Manual*. 2024. URL: <https://www.spice-space.org/spice-user-manual.html#spice-client> (besucht am 15.09.2024).
- [54] Branislav Sredojev, Dragan Samardzija und Dragan Posarac. "WebRTC technology overview and signaling solution design and implementation". In: *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE. 2015, S. 1006–1009.
- [55] Manuel Strauß-Aigner. *guacamole-sharp*. 2022. URL: <https://github.com/m-expunged/guacamole-sharp> (besucht am 19.08.2024).
- [56] K Swargiary. "Enhancing Educational Accessibility". In: (2024).
- [57] Michael Sweet und Ira McDonald. *Internet Printing Protocol/1.1: Model and Semantics*. RFC 8011. Jan. 2017. DOI: 10.17487/RFC8011. URL: <https://www.rfc-editor.org/info/rfc8011>.
- [58] taoyuan. *Generation of a Self Signed Certificate*. 2024. URL: <https://gist.github.com/taoyuan/39d9bc24bafc8cc45663683eae36eb1a> (besucht am 26.10.2024).
- [59] Teamviewer. *TeamViewer Web Client*. 2024. URL: <https://www.teamviewer.com/en/global/support/knowledge-base/teamviewer-classic/modules/web-client/> (besucht am 30.10.2024).
- [60] Ubuntuusers. *CUPS-PDF*. 2021. URL: <https://wiki.ubuntuusers.de/CUPS-PDF/> (besucht am 17.10.2024).
- [61] Ubuntuusers. *CUPS*. 2023. URL: <https://wiki.ubuntuusers.de/CUPS/> (besucht am 07.10.2024).

- [62] Wikipedia. *Lanczos resampling*. 2024. URL: [https://en.wikipedia.org/wiki/Lanczos\\_resampling](https://en.wikipedia.org/wiki/Lanczos_resampling) (besucht am 05.11.2024).