htlperg
innovativ | kreativ | praxisnah

RaspiSniffer

DIPLOMARBEIT

Höhere Abteilung für
Informatik
01/10/2024 – 04/04/2025

Projektmitglieder: Thomas Hintersteininger
Samuel Schlöglhofer
Daniel Kertesz

Betreuer:in: Prof. Dipl.-Ing. Helmut Otto

1 Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung der Arbeit haben wir das generative KI-Tool ChatGPT zu folgendem Zweck verwendet: Formulierung von Sätzen und Hilfeleistung beim Programmieren.

2 Gendererklärung

Im Interesse der besseren Lesbarkeit werden in dieser Diplomarbeit geschlechtsbezogene Bezeichnungen, die sich auf Frauen und Männer gleichermaßen beziehen, einheitlich in der im Deutschen üblichen maskulinen Form verwendet. Dies geschieht ausschließlich aus praktischen Gründen und impliziert weder eine Benachteiligung eines Geschlechts noch eine Verletzung des Gleichheitsgrundsatzes.

3 Danksagung

Wir möchten allen Personen der HTL Perg und der Kontron AG unseren herzlichen Dank aussprechen, die zum Gelingen dieser Diplomarbeit beigetragen haben.

Unser besonderer Dank gilt Herrn Professor Helmut Otto, der uns mit großem Engagement unterstützt hat.

Ein weiterer Dank geht an Herrn Christian Bimminger, durch dessen Unterstützung die Zusammenarbeit mit der Kontron AG möglich wurde.

Ebenso möchten wir uns herzlich beim gesamten Team der Kontron AG bedanken, die uns mit wertvoller Expertise beim technischen Teil der Arbeit tatkräftig zur Seite standen.

4 Abstract

The main idea of the RaspiSniffer is to monitor network connections in real time. The goal is to configure a Raspberry Pi to eavesdrop on network traffic, process it, and ultimately display the most essential information about the packets on a web interface.

The RaspiSniffer can be divided into three main components:

The Raspberry Pi is a compact and cost-effective single-board computer that runs a Unix-like operating system called Raspberry Pi OS. The Raspberry Pi allows us to eavesdrop on and record network traffic, either wired or wireless. The recordings are stored on the Raspberry Pi and can be transferred to a USB-stick or a connected PC. Additionally, required programs and services can be started automatically to automate the boot process.

The web interface provides the user with a simple interface on the Raspberry Pi and a connected device, where the core functions remain the same. However, the layout adjusts to the device to ensure ease of use. On the web interface, users have the ability to start monitoring and make adjustments for it. The saved recordings of the network traffic can then be deleted or transferred to a connected device or USB-stick. Furthermore, the web interface allows connection to a WLAN.

The backend acts as an interface between the other two components and contains most of the logic. It can control the Raspberry Pi by using commands and, through a web server, provide the web interface for the user. With the help of an API, the frontend can use API-calls to query and execute variables and methods from the backend.

5 Zusammenfassung

Die Grundidee des RaspiSniffer ist es, Netzwerkverbindungen in Echtzeit abzuhören. Das Ziel ist es, einen Raspberry Pi so zu konfigurieren, dass dieser den Datenverkehr in Netzwerken abhört, dann verarbeitet und schlussendlich auf einer Weboberfläche die notwendigsten Informationen über die Pakete angezeigt werden.

Der RaspiSniffer kann in drei Hauptkomponenten unterteilt werden:

Der Raspberry Pi ist ein kompakter und kostengünstiger Einplatinencomputer, auf dem ein Unix-ähnlichen Betriebssystem namens Raspberry Pi OS läuft. Der Raspberry Pi ermöglicht uns, den Datenverkehr in Netzwerken kabelgebunden oder kabellos abzuhören und aufzuzeichnen. Die Aufzeichnungen werden auf dem Raspberry Pi gespeichert und können auf einen USB-Stick oder einen verbundenen PC übertragen werden. Zusätzlich können benötigte Programme und Dienste automatisch gestartet werden, um den Startvorgang zu automatisieren.

Die Weboberfläche gibt dem Benutzer eine einfache Oberfläche auf dem Raspberry Pi und auf einem verbundenen Gerät, wobei sich die Kernfunktionen nicht unterscheiden. Das Layout passt sich aber an das Gerät an, damit es immer einfach handhabbar ist. Auf der Weboberfläche hat man die Möglichkeit, das Abhören zu starten und Anpassungen dafür vorzunehmen. Die gespeicherten Aufnahmen des Datenverkehrs können dann gelöscht oder auf ein verbundenes Gerät oder einen USB-Stick übertragen werden. Des Weiteren ermöglicht die Weboberfläche die Verbindung mit einem WLAN.

Das Backend dient als Schnittstelle zwischen den beiden anderen Komponenten und beinhaltet den Großteil der Logik. Mittels Befehlen kann es den Raspberry Pi steuern und durch einen Webserver kann es die Weboberfläche für den Benutzer bereitstellen. Mithilfe einer API kann das Frontend mit API-Calls Variablen und Methoden des Backends abfragen und ausführen.

Inhaltsverzeichnis

6 Einleitung

6.1 Auftraggeber

Der Auftraggeber der Arbeit war anfangs Secureguard, bis diese von der Kontron AG aufgekauft wurde und infolgedessen den Auftrag übernahm. Die Kontron AG ist ein führender Anbieter von intelligenten IoT-Lösungen. Das Unternehmen entwickelt und liefert innovative Technologien und Produkte, die in vielen Bereichen eingesetzt werden. Von automatisierten industriellen Abläufen und sicherem, intelligentem Transport bis zu fortschrittlichen Kommunikations-, Konnektivitäts-, Medizin-, Solar- und erneuerbaren Lösungen, die einen Mehrwert bieten. Das Angebot umfasst sowohl Hardware und Software als auch Dienstleistungen in der Elektronikfertigung. [1] [1]

Abbildung 1: Logo der Kontron AG
https://www.kontron.com/de

6.2 Motivation

Die fortschreitende Entwicklung der Netzwerktechnologie erfordert Werkzeuge zur Fehlerdiagnose, die sowohl effizient als auch flexibel einsetzbar sind. Aktuell verwendet die Kontron AG zur Analyse von Netzwerkproblemen Laptops mit Wireshark. Dieses Vorgehen gestaltet sich in der Praxis jedoch häufig als unhandlich und wenig flexibel.

Ziel dieser Diplomarbeit ist es, einen innovativen Lösungsansatz zur Unterstützung bei der Netzwerk-Fehlerbehandlung zu entwickeln. Im Mittelpunkt steht dabei der Einsatz eines Raspberry Pi, der als kompaktes, mobiles Diagnosewerkzeug dient. Das Konzept sieht vor, den Raspberry Pi so zu konfigurieren, dass er sowohl WLAN- als auch LAN-Verkehr erfassen kann, um Netzwerkprobleme gezielt und effizient zu identifizieren.

[1] *https://www.kontron.com/de/konzern/ueber-uns*

6.3 Ausgangslage

Secureguard gab den Auftrag, einen Raspberry Pi so zu konfigurieren, dass dieser in der Lage ist, den Datenverkehr in einem WLAN oder einem LAN abzuhören und Aufzeichnungen zu erstellen, damit diese für Debugging-Zwecke genutzt werden können. Dieser Raspberry Pi soll eine simple Benutzeroberfläche hosten, die neben dem Anzeigen des Datenverkehrs weitere Funktionalitäten bieten soll. Dies soll auf einem kleinen 4:3-Bildschirm oder einem größeren 16:9-Bildschirm möglich sein. Das Speichern der Aufzeichnungen auf dem Gerät und USB-Stick soll auch gegeben sein.

Für das Projekt wurden ein Raspberry Pi, eine SD-Karte, ein USB-Stick, zwei USB-Ethernet-Adapter, ein temporärer Bildschirm und benötigte Strom- und HDMI-Kabel bereitgestellt.

Die technischen Vorgaben waren das Verwenden eines ASP.NET Core-Backends und das Erstellen einer Blazor-Webseite.

6.4 Projektinhalt

Das Projekt kann in drei Hauptkategorien eingeteilt werden. Die Hardware und das Betriebssystem, das Backend und das Frontend.

6.4.1 Hardware und Betriebssystem

Das Thema Hardware und Betriebssystem beinhaltet die Konfiguration eines Raspberry Pis. Dieser soll mitsamt SD-Karte, Bildschirm und USB-Ethernet-Adapter ausgestattet werden. Er soll über diverse Skripte und Einstellungen Folgendes ermöglichen:

- Abhören und Aufzeichnen von Paketen einer Netzwerkverbindung.
- Der Raspberry Pi soll den Datenverkehr nicht behindern, somit soll dieser als Bridge fungieren, wenn der Raspberry Pi zwischen zwei oder mehr Geräten hängt.
- Das Backend und der Webserver sollen beim Start des Raspberry Pi automatisch gestartet werden. Weiter soll die Webseite im Vollbildmodus geöffnet werden.
- USB-Sticks sollen, wenn diese angeschlossen werden, automatisch gemountet werden.
- Ethernet-Adapter sollen beim Anschließen richtig konfiguriert werden.
- WLAN-Verbindungen sollen durch Scripte aufgebaut werden.

6.4.2 Backend

Das Thema Backend umfasst die Datenverarbeitung und die Datenbereitstellung. Dabei sollen die verarbeiteten Daten über eine API für das Frontend bereitgestellt werden. Die Backend-API soll Folgendes bereitstellen:

- Die Möglichkeit, das Abhören von Netzwerken mit diversen Filtern starten zu können. Diese sollen dann auch als File auf dem Raspberry Pi abgespeichert werden.
- Die abgehörten Pakete sollen in bearbeiteter Form abgefragt werden können.
- Jegliche Funktionen um die WLAN-Verbindung anpassen zu können.
- Die auf dem Raspberry Pi gespeicherten Aufzeichnungen sollen abgerufen, heruntergeladen und gelöscht werden können.
- Zusätzlich wird eine Funktion implementiert, um die erfassten Dateien auf einen USB-Stick zu verschieben und diesen anschließend sicher auszuhängen.

6.4.3 Frontend

Das Thema Frontend beinhaltet die Erstellung der Benutzeroberfläche. Diese soll für 4:3- und 16:9-Bildschirme leicht abgeänderte Versionen anzeigen. Die Webseite selbst soll in drei Seiten aufgeteilt werden.

- Die Hauptseite: Auf der Hauptseite soll der Datenverkehr des aufgezeichneten Netzwerks angezeigt werden. Weiter sollen Filter für die Aufzeichnungen ausgewählt werden können. Die Aufzeichnungen sollen auf dieser Seite gestartet werden. Von dieser Seite soll man auch auf die Anderen zugreifen können.
- Die Recordings-Seite: Auf der Recordings-Seite sollen die abgeschlossenen Aufzeichnungen die auf dem Raspberry Pi gespeichert sind, angezeigt werden. Diese sollen dann auf einem anderen Gerät heruntergeladen oder auf einen USB-Stick gespeichert werden können. Weiter sollen nicht benötigte Dateien gelöscht werden können. Ein Button für das Trennen eines USB-Sticks soll auch gegeben sein. Letztendlich soll es einen Button geben der den Benutzer zurück zur Hauptseite führt.
- Die WIFI-Seite: Auf der WIFI-Seite sollen alle möglichen Internetverbindungen angezeigt werden. Es soll eine kleine Box existieren, in der Informationen über die Netzwerkverbindung des Raspberry Pi stehen. Über einen Button soll man sich mit einem WLAN verbinden können und mit einem anderen soll man sich von der derzeitigen WLAN-

Verbindung trennen können. Letztendlich soll es einen Button geben der den Benutzer zurück zur Hauptseite führt.

Schlussendlich soll das Frontend die vom Backend bereitgestellte API verwenden.

6.5 Projektumfeld

6.5.1 Projektteam

Das Projektteam besteht aus Thomas Hintersteininger, Daniel Kertesz und Samuel Schlöglhofer, alle drei Schüler der Höheren Technischen Lehranstalt Perg. Thomas Hintersteininger hat schon viel Erfahrung gesammelt im Frontend mit Blazor durch sein Praktikum bei SecureGuard, welches von der Kontron AG übernommen wurde. Samuel Schlöglhofer hatte schon immer viel Freude, wenn es darum ging, ein Backend aufzubauen. Daniel Kertesz hat schon immer Interesse am Raspberry Pi gezeigt und war dann auch für die Skripte auf dem Raspberry Pi zuständig.

Abbildung 2: Samuel Schlöglhofer (links), Thomas Hintersteininger (mitte) und Daniel Kertesz (rechts)

6.5.2 Projektbetreuer

Die Diplomarbeit wurde im technischen und theoretischen Teil durch Dipl.-Ing. Helmut Otto betreut. Er hat das Projektteam durch sein Wissen im Bereich ASP.NET unterstützen können.

6.5.3 Vorgehensmodell

Bei dem praktischen Teil der Diplomarbeit wurde das hybride Vorgehensmodell Scrumban verwendet. Scrumban ist eine Kombination aus Scrum und Kanban und wurde entwickelt, um die Vorteile beider Vorgehensmodelle zu nutzen und eine flexible Arbeitsweise zu ermöglichen. Scrum basiert auf festen Sprints, definiert Rollen und ein Sprint-Backlog. Kanban setzt auf einen kontinuierlichen Arbeitsfluss mit Work-in-Progress(WIP)-Limits und einer visuellen Darstellung der Aufgaben in einem Kanban-Board.

Bei Scrumbun werden die Daily Standups von Scrum beibehalten, aber der feste Sprint-Backlog verworfen. Stattdessen wird das Kanban-Board verwendet, wo Aufgaben nach Priorität und Verfügbarkeit den Teammitgliedern zugeteilt werden.

6.6 Aufbau und Struktur

6.6.1 Aufbau Rasperry Pi

Der wichtigste Teil des RaspiSniffers sind die verschiedenen Anschlüsse, hier Ports genannt. Der native Ethernet Port, das ist jener, der auf einem normalen Raspberry Pi montiert ist, muss immer mit dem Netzwerk verbunden sein, über das man die Bedienungsoberfläche aufrufen will. Über diesen Port werden die Website und das Backend gehostet. Außerdem muss man an den Micro-HDMI Port ein Display anschließen. Der Bildschirm wird benötigt, um die Website lokal auf dem Raspberry Pi anzuzeigen. Der letzte Bestandteil des RaspiSniffers sind die USB-Ethernet Adapter. Damit der RaspiSniffer eine Kabelverbindung überwachen kann, benötigt er mindestens zwei dieser Adapter. Das Gerät sollte durch einen Adapter mit dem Computer verbunden sein, dessen Verbindung überprüft werden soll. Der andere Adapter muss an das Netzwerk angeschlossen sein, mit dem sich der Computer verbinden will. Es können auch mehrere USB-Ethernet Adapter verwendet werden. In diesem Fall fungiert der Raspberry Pi als Switch, der den Netzwerk Traffic zwischen allen angeschlossenen Geräten aufzeichnet.

6.6.2 Schnittstellenarchitektur

Die Datenübertragung zwischen dem Frontend und dem Backend läuft über eine API. Über verschiedene Endpunkte in dieser API kann das Frontend bestimmte Befehle auf dem Raspberry Pi ausführen, und die daraus entstandenen Daten abfragen. Alle Daten werden in einem JSON-Format über HTTP übertragen. Die Übertragung ist nicht verschlüsselt, da der RaspiSniffer nur netzwerkintern verwendet werden soll, und keine sensitiven Daten übermittelt werden.

6.6.3 Aufbau Frontend

Das Frontend besteht aus zwei Websites mit kleinen Unterschieden. Eine Website wurde für einen PC-Bildschirm entwickelt, die andere für einen kleineren Bildschirm, der direkt am Raspberry Pi angebracht ist. Beide Websites erledigen großteils trotz verschiedenen Layouts dieselben Funktionen. Eines der Features, die nur auf einer Version der Website vorhanden ist, ist die Download Funktion. Diese ist nur in der Website verfügbar, die im Netzwerk gehostet ist, da

die Files, die zum Download zur Verfügung stehen auf dem Raspberry Pi schon vorhanden sind. Die Hauptfunktion der Website ist es, den sniffing Vorgang zu starten und zu stoppen. Diese Funktion befindet sich auf der Startseite. Die anderen Funktionen des Frontends sind auf Pop-Ups und verschiedene Seiten verteilt.

7 Grundlagen und Methoden

7.1 Fachbegriffe

Im Laufe der Diplomarbeit werden einige Fachbegriffe der Netzwerktechnik verwendet. Diese werden hier erklärt, um Verwirrung zu vermeiden.

7.1.1 Packet

Das Packet, zu Deutsch Datenpaket, ist ein kleiner Teil einer größeren Nachricht. Wenn Daten über ein Netzwerk versendet werden, werden sie in Packets aufgeteilt, und diese werden dann einzeln verschickt. Der Empfänger der Daten kann aus allen erhaltenen Paketen die verschickten Daten wieder zusammensetzten. Ein Packet besteht aus dem Header und dem Payload, letzterer wird oft Data genannt. Der Payload beinhaltet die Daten, die der User verschicken will. Der Header speichert die verschiedenen Attribute des Datenpakets. Die Felder, die für diese Diplomarbeit besonders wichtig sind, sind das sogenannte Transport Attribut und das Destination IP Address Attribut3. Transport steht für das Transport Protokoll, meistens UDP oder TCP, mit dem das Paket versendet wird. Das Destination IP Address Attribut beinhaltet die IPv4 Adresse, an die das Datenpaket adressiert ist. Diese zwei Attribute können einem Netzwerkadministrator sehr hilfreich sein, falls ein unerklärlicher Fehler im Netzwerk auftritt.[2] [2]

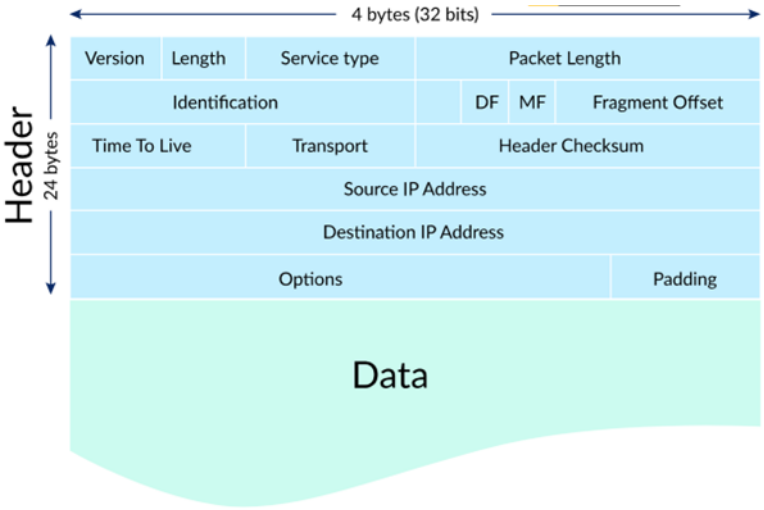

Abbildung 3: Skizze eines IP Packet
https://www.khanacademy.org/computing/ap-computer-science-principles/the-internet/x2d2f703b37b450a3:routing-with-redundancy/a/ip-packets

[2] *https://www.cloudflare.com/learning/network-layer/what-is-a-packet/*

7.1.2 Man in the Middle

Ein Man-in-the-Middle-Angriff, kurz MITM-Angriff, ist ein Hacking Angriff, dessen Ziel es ist, einen Datenaustausch zwischen zwei Usern abhören, verändern, oder stoppen zu können[3] [3]. Meistens erreicht der Angreifer dieses Ziel, indem er sich als Teil des Gesprächs ausgibt, wodurch der andere Gesprächspartner dessen Daten an den Angreifer schickt. Falls der Angreifer diesen Schritt vollzieht, kann er die abgefangenen Daten lesen, und falls er will, leicht verändert an den originalen Gesprächspartner weiterschicken. Ein MITM-Angriff ist sehr lukrativ, da herkömmliche Verschlüsselungen in diesem Angriff keine Rolle spielt. Der Angreifer wird als beabsichtigter Empfänger der Daten gesehen und kann daher die versendeten Daten entschlüsseln.[4] [4]. Dieser Angriff ist für diese Diplomarbeit relevant, da sie einem MITM-Angriff ähnlich ist. Der Raspberry Pi kann, wie in dem Angriff, den Datenaustausch zwischen zwei Usern aufzeichnen. Der große Unterschied zwischen dem RasPiSniffer und einem MITM-Angriff ist die Verschlüsselung. Der RasPiSniffer kann den Payload eines Packets, also die Daten, die verschickt werden, nicht entschlüsseln. Er kann nur den Header des Datenpakets entziffern, und deshalb nur manche Details der einzelnen Packets aufzeichnen.

7.1.3 Singleton

Der Singleton ist ein Designmuster, oft auch Design Pattern genannt, das für diese Diplomarbeit sehr wichtig ist. Die Grundidee des Singleton ist es, zu garantieren, dass von einer Klasse nur maximal ein Objekt instanziiert werden kann [5][5]. Somit kann das gesamte Programm auf die gleiche Instanz der Klasse zugreifen. Dieses Designmuster ist sehr nützlich für Klassen, die Daten speichern, die im gesamten Programm zur Verfügung stehen sollten. Außerdem kann es von verschiedenen Threads angesteuert werden. Deswegen ist das Singleton einer der einfachsten Wege, mehrere Threads auf die gleichen Daten zugreifen zu lassen. In diesem Fall könnte in extremen Fällen der Singleton einen Engpass darstellen, da immer nur ein Thread auf einmal auf einen Singleton zugreifen kann[6][6]. In dieser Diplomarbeit werden jedoch nur wenige Threads auf einmal verwendet, und Zugriffe auf die Daten des Singleton halten sich bezüglich Dauer und Häufigkeit in Grenzen.

[3] *https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/*
[4] *https://www.fortinet.com/de/resources/cyberglossary/man-in-the-middle-attack*
[5] *https://refactoring.guru/design-patterns/singleton*
[6] *ttps://medium.com/@berktorun.dev/navigating-the-complex-world-of-singletons-in-multithreaded-applications-867bb03eed02*

7.2 Verwendete Technologien

7.2.1 C#

Abbildung 4: C# Logo

Da uns vom Auftraggeber Kontron AG vorgeschrieben wurde, das Projekt mit ASP.NET Core umzusetzen, war die Auswahl der Programmiersprache sehr einfach. C# (ausgesprochen C-Sharp) ist eine objektorientierte, typsichere, Open-Source-Programmiersprache. Open-Source bedeutet, dass der Quellcode einer Software öffentlich zugänglich ist, von jedem eingesehen, genutzt, verändert und weiterverbreitet werden kann. Sie wurde von Microsoft als Teil des .NET Frameworks entwickelt und ist eine sehr stark vertretene Programmiersprache bei Desktop- , Web-Anwendungen, mobilen Apps und Videospielen.[7] [7]

7.2.2 Asp.Net Core

Abbildung 5: ASP.NET Core Logo

ASP.NET Core ist ein plattformunabhängiges, leistungsfähiges und modulares Webframework. Es wurde von Microsoft als Weiterentwicklung von ASP.NET erstellt, um Webanwendungen und APIs effizient und flexibel zu erstellen. Die Eigenschaft der Plattformunabhängigkeit von ASP.NET Core war für unser Projekt wichtig, da alles unter Linux laufen muss und nicht unter Windows. Zudem unterstützt ASP.NET Core Multithreading, was eine effiziente Verarbeitung mehrerer Aufgaben gleichzeitig ermöglicht. Ein Thread ist ein einzelner Ausführungsstrang innerhalb eines Programms, der unabhängig arbeitet. Multithreaded bedeutet, dass ein Programm mehrere Threads gleichzeitig ausführt, um Aufgaben parallel zu verarbeiten. Dadurch kann es effizienter arbeiten, indem es z. B. auf mehreren CPU-Kernen läuft oder gleichzeitig auf Eingaben wartet und Berechnungen durchführt.

[8][8]

7.2.3 Blazor

Abbildung 6: Blazor Logo

Blazor ist ein Webframeework, welches von Microsoft als Teil von ASP.NET Core entwickelt wurde. Es ist eine Weiterentwicklung von Razor, welches ein Teil von ASP.NET ist. Blazor bietet

[7] *https://dotnet.microsoft.com/en-us/languages/csharp*
[8] *https://dotnet.microsoft.com/en-us/apps/aspnet*

eine gute Alternative für Webentwickler, um eine Single-Page Application oder Web-Frontends zu erstellen, die im .NET-Ökosystem bleiben wollen und nicht JavaScript verwenden wollen. Blazor kann man auf zwei verschiedenen Arten hosten, als Web-Assembly und Blazor Server. Bei Web-Assembly wird die ganze Logik im Browser ausgeführt statt auf dem Server. Blazor Server hingegen ist eine eher klassischere Art fürs Hosten, denn die ganze Logik wird am Server ausgeführt und die UI werden an den Client geschickt. Für unser Projekt, wo der Hoster nicht viel Leistung hat, eignet sich Web-Assembly perfekt.[9] [9]

7.2.4 Raspberry Pi

Ein Raspberry Pi (auch RasPi genannt) ist ein kostengünstiger Einplatinencomputer, der ungefähr so groß ist wie eine Kreditkarte. Originell wurde er entwickelt, um das Programmieren und Informatikkenntnisse in Schulen und Entwicklungsländern zu fördern. Es hat sich aber schnell herausgestellt, dass man einen RasPi vielseitig einsetzen kann, z.B.:

Abbildung 7: Raspberry Pi Logo

- Bildung: Um den Schülern Programmieren und Informatik beizubringen
- Bastelprojekte: RasPis haben auch im privaten Bereich viele Einsätze, z.B.:
  - Heimautomatiesierung
  - Wetterstation
  - Retro-Gaming-Konsolen
- Server: Als kostengünstige Webserver, Medienserver oder Cloud-Speicherlösung
- IoT Projekte: Da er sich leicht mit Sensoren und anderen Geräten verbinden lässt
- Professionelle Anwendung: Einige Unternehmen verwenden RasPis auch zur digitalen Beschilderung, Kiosksysteme oder zur industriellen Automatisierung

[10][10] Seit der Veröffentlichung des Raspberry Pi im Jahr 2012 wurden viele verschiedene Modelle mit verschiedenen Ausstattungen entwickelt.

- Raspberry Pi 1:

[9] *https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor*
[10] *https://www.raspberrypi.com/*

Abbildung 8: Raspberry Pi 1

Der Raspberry Pi 1 war der erste Raspberry Pi, der 2012 auf den Markt gebracht wurde. Im Jahr 2014 wurden diese dann mit mehr RAM aufgerüstet.

- Modell A: 1x , , 256 MB RAM
- Modell A+: 1x , , 256 MB RAM, ab 2014 512 MB RAM
- Modell B: Ethernet, 2x , , 256 MB RAM, ab 2014 512 MB RAM
- Modell B+: Ethernet, 4x , , 256 MB RAM, ab 2014 512 MB RAM

- Raspberry Pi 2:

Abbildung 9: Raspberry Pi 2

Der Nachfolger vom Raspberry Pi 1 wurde im Jahr 2015 mit einer besseren CPU und mehr RAM veröffentlicht. Dabei gab es nur ein Modell. Im Jahr 2016 wurde die CPU verbessert.

- Modell B: Ethernet, 4x USB 2.0, HDMI, 1 GB RAM

- Raspberry Pi Zero:

Abbildung 10: Raspberry Pi Zero

Der RasPi Zero ist ungefähr halb so groß wie die bisherigen Raspberry Pis. Von dieser Version wurden seit 2015 drei verschiedene Modelle entwickelt.

  - Zero: 1x Micro USB, Mini-HDMI, 512 MB RAM, Ohne jeglicher Wireless Technologie (WLAN, Bluetooth)
  - Zero W: 1x Micro USB, Mini-HDMI, 512 MB RAM, Wireless Technologien wurden hinzugefügt (WLAN, Bluetooth)
  - Zero WH: 1x Micro USB, Mini-HDMI, 512 MB RAM, GPiO-Header schon verlötet
  - Zero 2 W: 1x Micro USB, Mini-HDMI, 512 MB RAM, Ähnlich wie der Zero W

- Raspberry Pi 3:

Abbildung 11: Raspberry Pi 3

Der Raspberry Pi 3 ist eine weitere Verbesserung zum Raspberry Pi 2 aus dem Jahr 2016. In den RasPi 3 wurde eine bessere CPU und in manchen Modellen auch ein, Ethernet-Anschluss verbaut.

  - Modell A+: 1x USB 2.0, HDMI, 1 GB RAM
  - Modell B: Ethernet, 4x USB 2.0, HDMI, 1 GB RAM
  - Modell B+: Ethernet, 4x USB 2.0, HDMI, 1 GB RAM

- Raspberry Pi 4 /400:

Abbildung 12: Raspberry Pi 4

Der Raspberry Pi 4 ist die Weiterentwicklung aus 2019. Dabei wurde nur ein einziges Modell veröffentlicht. Dieser war durch die Micro--Anschlüsse in der Lage, zwei Bildschirme mit einer Auflösung von bis zu 4K zu betreiben.

  - o Modell B, Ethernet, 2x USB 2.0, 2x USB 3.0, 2x Micro-HDMI mit einer Auflösung bis zu 4k; 1, 2, 4 oder 8 GB RAM, wobei die mit 1 GB RAM nicht mehr produziert, werden

- Raspberry Pi 5:

Abbildung 13: Raspberry Pi 5

Der Raspberry Pi 5 ist die neueste Version, welche im Jahr 2023 auf den Markt gebracht wurde. Zurzeit gibt es nur ein Modell. Dieses soll auch 2 mal so stark sein wie der Raspberry Pi 4.

  - Modell Standard: Ethernet, 2x USB 2.0, 2x USB 3.0, 2x Micro-HDMI, 4 oder 8 GB RAM

[11][11]

Um das am besten geeignete Raspberry Pi Modell für das Projekt zu bestimmen, wurden die derzeit auf dem Markt verfügbaren Modelle analysiert. Dabei wurden der Raspberry Pi 4 und der Raspberry Pi 5 genauer betrachtet. Die Entscheidung fiel darauf, dass der Raspberry Pi 4 für die Anforderungen des Projekts vollkommen ausreicht.

Anschließend wurde untersucht, in welchen Ausstattungsvarianten der Raspberry Pi 4 erhältlich ist. Dabei wurde festgestellt, dass vier unterschiedliche Varianten existieren, die sich lediglich in

[11] *https://de.wikipedia.org/wiki/RaspberryPiRaspberry − Pi − Modelle*

der Größe des Arbeitsspeichers (RAM) unterscheiden: 1 GB, 2 GB, 4 GB und 8 GB. Da auf dem Raspberry Pi ein Webserver gehostet wird, der Datenverkehr mitgehört und verarbeitet werden muss, wurde der Raspberry Pi 4 mit 8 GB RAM als die einzige sinnvolle Option identifiziert.

7.2.5 Raspberry Pi OS

Raspberry Pi Foundation hat auch ein eigenes Betriebssystem für die Raspberry Pis entwickelt. Raspberry Pi OS, früher Raspbian, welches eine auf Debian basierende Linux-Distribution ist. Man kann auch andere Betriebssysteme installieren, aber Raspberry Pi OS ist für die Raspberry Pis optimiert. [12][12]

Abbildung 14: Raspberry Pi OS Logo

7.2.6 Bash Scripting Language

Bash ist eine Unix-Shell und Scriptsprache, die in den meisten Linux-Distributionen als Standard-Shell verwendet wird. Sie wurde 1989 als Alternative zur Bourne Shell (sh) entwickelt und bietet zahlreiche Erweiterungen wie Arrays und eine verbesserte Kontrollstruktur. Sie ermöglicht die Automatisierung von Aufgaben über Skripte und die Verwaltung von Systemprozessen. [13][13]

Abbildung 15: Bash Logo

7.2.7 Sniffertools

Um das geeignete Sniffer-Tool für die Netzwerkanalyse auszuwählen, wurden verschiedene Optionen intensiv untersucht. Der Fokus lag auf vier Tools: tcpdump, Sniffnet, netsniff-ng und slsnif. Jedes Tool wurde daraufhin überprüft, inwieweit es den Anforderungen entspricht, welche Funktionen es bietet und wie gut es sich integrieren lässt.

[12] *https://de.wikipedia.org/wiki/Raspberry_Pi#Raspberry_Pi_OS*
[13] *https://www.gnu.org/software/bash/*

- tcpdump:

  Ein leistungsstarkes und vielseitiges Tool, das über die Kommandozeile gesteuert wird. Es unterstützt die Filterung des Netzwerkverkehrs und ermöglicht die Speicherung der Daten in Dateien, die später in Tools wie Wireshark analysiert werden können. Es zeichnet sich durch einfache Bedienbarkeit und breite Unterstützung aus. [14][14]

- Sniffnet:

  Dieses Tool bietet eine grafische Benutzeroberfläche, was die Integration in das Projekt erschwerte, da eine eigene Benutzeroberfläche erstellt werden sollte. Aus diesem Grund wurde es ausgeschlossen. [15][15]

- netsniff-ng:

  Neben den Funktionen von tcpdump nutzt es Zero-Copy-Mechanismen, um die CPU-Auslastung zu reduzieren. Dies erhöht die Effizienz und Leistungsfähigkeit. Allerdings bestehen Einschränkungen bei der Zero-Copy-Kompatibilität mit Raspberry-Pi-Hardware, weshalb der Einsatz externer leistungsstarker Ethernet-Adapter erforderlich wäre.[16][16]

  - Zero-Copy Receive-Ringe: Bei traditionellen Netzwerk-Datenübertragungen werden die Datenpakete mehrmals kopiert, bevor sie die Anwendung erreichen, was zu einer höheren CPU-Auslastung führt. Zero Copy versucht, diese Kopiervorgänge zu eliminieren oder zu reduzieren, indem die Datenpakete direkt von der Netzwerkkarte zur Anwendung übertragen werden. Receive-Ringe sind die Datenstrukturen, die Netzwerkkarten zur Verwaltung der Pakete verwenden. Diese werden in einem Ring-Puffer abgelegt, bevor sie vom System weiterverarbeitet werden. Durch die Kombination dieser Techniken werden empfangene Pakete direkt in den Speicherbereich der Anwendung abgelegt, ohne dass sie zuvor zwischen mehreren Pufferzonen verschoben werden. Dafür ist eine enge Integration zwischen der Netzwerkkarte, dem Betriebssystem und der Anwendung erforderlich. [17][17][18][18]

- slsniff:

  Dieses Tool ist auf Echtzeitanalysen spezialisiert, bietet jedoch keine Möglichkeit, Daten direkt für Wireshark zu speichern, was zusätzliche Schritte erforderlich gemacht hätte. [19][19]

[14] *https://www.tcpdump.org/*
[15] *https://www.tcpdump.org/*
[16] *http://netsniff-ng.org/*
[17] *https://en.wikipedia.org/wiki/Zero-copy*
[18] *https://pdos.csail.mit.edu/6.828/2021/labs/net.html*
[19] *https://linux.die.net/man/1/slsnif*

Um die beste Wahl für das Projekt zu treffen, wurde eine gründliche Abwägung durchgeführt. Dabei erwies sich netsniff-ng als das am besten geeignete Tool, da es neben den Standardfunktionen auch eine hohe Effizienz und Flexibilität bietet. Die einzige Einschränkung bestand in der Hardware-Kompatibilität. Da netsniff-ng jedoch selbst ohne Zero-Copy noch Vorteile gegenüber tcpdump bietet, wurde es als die beste Option ausgewählt.

7.2.8 Nginx

Nginx ist ein leistungsstarker und ressourcenschonender Webserver und Reverse-Proxy, der ursprünglich von Igor Sysoev entwickelt wurde. Im Gegensatz zu Apache ist Nginx effizienter, wenn es darum geht, mehrere Verbindungen gleichzeitig zu behandeln. Nginx wird dadurch bei eher leistungsschwächeren Servern in Einsatz genommen. Ein weiterer großer Verwendungszweck von Nginx ist als Reverse-Proxy. Ein Reverse-Proxy ist ein Server, der Anfragen von Benutzern entgegennimmt und auf einen oder mehrere Backend-Server weiterleitet. Dadurch wird die Last, die ein Server tragen muss, auf eine Vielzahl von Servern aufgeteilt.

Abbildung 16: Nginx Logo

Für dieses Projekt wurde Nginx gewählt weil es einfacher ist einen Nginx Webserver zu installieren und Konfigurieren. [20][20]

7.3 Verwendete Entwicklungsumgebungen

7.3.1 JetBrains Rider

JetBrains Rider ist eine integrierte Entwicklungsumgebung (IDE) für .NET, C# und F#, die von JetBrains entwickelt wurde. Sie basiert auf der IntelliJ-Plattform und kombiniert leistungsstarke Code-Analyse und Debugging-Funktionen von ReSharper[21][21]. Im Vergleich zu Visual Studio Code zeichnet sich Rider durch eine schnelle Performance, tiefgehende Code-Analyse und Cross-Plattform-Unterstützung aus. [22][22]

Abbildung 17: JetBrains Rider Logo

7.3.2 Nano

[20] *https://nginx.org/*
[21] *https://www.jetbrains.com/resharper/*
[22] *https://www.jetbrains.com/rider/*

GNU Nano (oder einfach "Nano") ist ein einfacher Texteditor, der über die Kommandozeile (CLI) von Unix- und Linux-Systemen genutzt wird. Er ist standardmäßig in den meisten Linux-Distributionen und auf Raspberry Pi OS vorinstalliert. Mit Nano kann man einfach Dateien bearbeiten. Es ist sehr ähnlich zu dem Editor/Notepad aus Windows. [23][23]

Abbildung 18: Nano Logo

7.4 Verwendete Bibliotheken

Im Laufe dieser Diplomarbeit wurden verschiedene öffentliche Bibliotheken verwendet. Die wichtigste und interessanteste Bibliothek wird hier genauer erklärt.

7.4.1 CLI.wrap

CLI.wrap[24][24] ist eine Bibliothek, die es ermöglicht, in einem C# ProgRAMm beliebige Command Line Befehle zu erstellen und auszuführen. Sie bietet einige Features, die vergleichbare Bibliotheken nicht haben. Außerdem ist sie einfach zu verstehen. Wenn das C# Programm in Linux ausgeführt wird, führt es die Befehle in Linux Shell aus, statt in der Windows CMD. Die wichtigste Klasse der Bibliothek ist die Command Klasse. Diese Klasse speichert den Befehl, der ausgeführt werden soll, und sie ermöglicht es, Argumente zu dem Befehl hinzuzufügen. Die Ausgabe des Befehls kann durch verschiedene Methoden verarbeitet werden. Die Methode, die in dieser Diplomarbeit am meisten verwendet wird, ist „PipeTarget.ToDelegate()“. Diese Funktion erlaubt es, ein Func Objekt mit den richtigen Parametern als Handler für die Ausgabe des Befehls zu verwenden. Der Vorteil dieser Methode ist, dass die Methode sofort mit der Ausgabe aufgerufen wird, wenn der Befehl eine Zeile ausgibt. Eine andere Herangehensweise ist „PipeTarget.ToStringBuilder()“. Diese Methode gibt nur, nachdem der Befehl fertig ist, die Ausgabe an den StringBuilder weiter. Der Vorteil dieser Herangehensweise ist, dass sie sehr einfach zu implementieren ist und keine callback Methoden benötigt.

[23] *https://www.nano-editor.org/*
[24] *https://github.com/Tyrrrz/CliWrap*

7.5 Verwendete Hardware

7.5.1 Raspberry Pi

Damit die Diplomarbeit klein, kompakt und einfach zum Transportieren ist, haben wir den Einplatinencomputer namens Raspberry Pi verwendet. Dieser hat das Blazor WebAssembly Frontend und das Backend gehostet. (Genauer in §7.2.4 Raspberry Pi erklärt)

7.5.2 Bildschirm

Damit man nicht immer auf einem externen PC auf die Website zugreifen muss, hat der Raspberry Pi einen Bildschirm mit einem Seitenverhältnis von 4:3.

7.5.3 USB-to-LAN Adapter

Die USB-to-Lan-Adapter wurden an den Raspberry Pi angeschlossen, damit dieser mehrere Ethernet-Anschlüsse hat. Diese werden auch verwendet, um den Netzwerkverkehr abzuhören. Im weiteren Verlauf werden sie an manchen Stellen auch Ethernet-Anschluss oder Ethernet-Adapter genannt.

7.5.4 USB-Stick

Der USB-Stick wurde verwendet, um die von netsniff-ng gespeicherten Sniffing-Aufnahmen auf diesem zu speichern, um diese an einem anderen PC mit Wireshark analysieren zu können.

8 Implementierung

8.1 Raspberry Pi

8.1.1 Raspberry Pi Grundlagen

Damit der Raspberry Pi überhaupt verwendet werden kann, muss zuerst Raspberry Pi OS installiert werden. Da der Raspberry Pi keinen internen Speicher hat, muss das Betriebssystem auf eine SD-Karte geschrieben werden. Da es keine vorinstallierte Version vom Betriebssystem gibt wie bei Windows, muss es aus dem Internet von der Homepage von Raspberry Pi heruntergeladen werden. Wenn man diesen Installer ausführt, kann man dann auf die SD-Karte Raspberry Pi OS installieren. Ein Installer ist ein Programm, welches ein anderes Programm installiert und Einstellungen vornimmt. Der Installer bietet auch die Möglichkeit, andere Betriebssysteme zu installieren. Darunter fallen mehrere Versionen von Raspberry Pi OS, nämlich die 64-bit, 32-bit und die 32-bit Legacy-Versionen. Andere mögliche Betriebssysteme sind jede mögliche Art von Linux-Distributionen wie Ubuntu, Apertis oder Kali Linux. In dem offiziellen Online-Shop von Raspberry Pi, BerryBase[25][25], sind weitere mögliche Betriebssysteme aufgelistet.

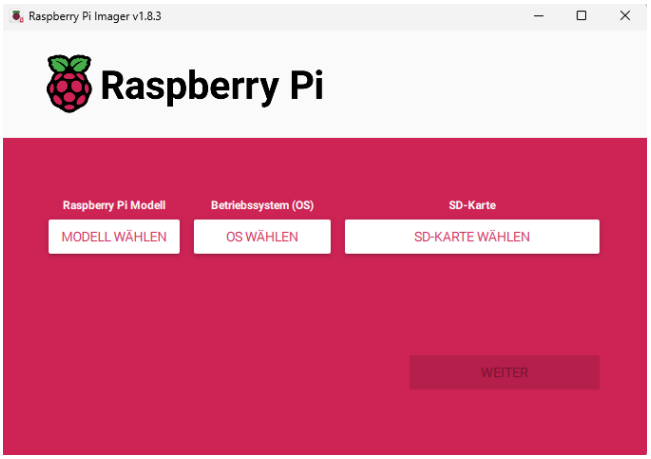

Abbildung 19: Raspberry Pi OS Installer

Nachdem Raspberry Pi OS auf der SD-Karte installiert wurde, kann endlich der Raspberry Pi verwendet werden. Bei diesem Projekt wurde die Version 5.3 von Raspberry Pi OS (64 bit) verwendet. Auf diesem muss dann SSH (Secure Shell, eine Technologie, mit der man andere

[25] *https://blog.berrybase.de/raspberry-pi-die-wichtigsten-7-betriebssysteme-im-vergleich/*

Geräte über die Konsole fernsteuern kann) aktiviert werden. Weiterhin müssen die ganzen Pakete installiert werden. Wie und welche Pakete installiert werden müssen, wird in den jeweiligen Kapiteln beschrieben.

8.1.2 Netzwerk- und Sniffer-Tools

Implementierung und Tests von Netzwerkanalysen

Nachdem die Entscheidung für netsniff-ng gefallen war, begann die praktische Umsetzung. Zunächst wurde das Tool auf dem Raspberry Pi installiert. Dies geht mit "sudo apt install netsniff-ng". Damit die Funktionalität von netsniff-ng effektiv vom Backend gesteuert werden konnte, mussten erstmals mehrere Test-Skripte erstellt werden. Diese sehen wie folgt aus.

Listing 1: Netsniff-ng Test Skript

```
1       netsniff-ng -i br0 -o lan_capture.pcap --filter "tcp port 80"
2
3       netsniff-ng -i wifi0 -o wifi_capture.pcap --filter "udp port 53"
```

In diesen Beispielen wird mit dem "`-i`" die Netzwerkschnittstelle, die "abgehört" werden soll, definiert, im Falle des ersten Beispiels ist es "br0", also die Bridge, die von uns definiert wurde (siehe §8.1.3 Raspberry Pi als Bridge einrichten) und im zweiten wurde dann die Schnittstelle "wifi0", welche ganz normal WLAN ist. Mit dem "`-o`" wird definiert, dass alle abgefangenen Pakete in eine .pcap geschrieben werden. .pcap Dateien ermöglichen die spätere Analyse der Aufnahme mit Wireshark. Nach dem "`-o`" kann auch ein Pfad angegeben werden, wo diese .pcap Datei gespeichert werden soll z.B.: "/media/RaspiSniffer/USB/lan_capture.pcap". Mit dem "`-filter`" kann man nach bestimmten Paketen filtern. Dabei kann man nach den verschiedenen Transportprotokollen filtern (z.B.: TCP, UDP, sctp, dccp). Eine weitere Filtermöglichkeit ist nach dem Port, dies geht wie in den zwei Beispielen gezeigt wird mit "`-filter port 80`", in diesem Fall werden alle Pakete abgehört, die durch den Port 80 verschickt werden.

Nachdem für das Projekt alle nötigen Verwendungen von netsniff-ng getestet wurden, wurde im Backend netsniff-ng mit CLI.Wrap eingebunden und verwendet.

8.1.3 Konfiguration und Skripte

Raspberry Pi als Bridge einrichten

Der Raspberry Pi hat einen OnBoard LAN Anschluss, welcher verwendet wird, um auf die Weboberfläche zu kommen. Damit dieser Anschluss nur für die Weboberfläche da ist, muss das Netzwerk-Interface (Ein Netzwerk-Interface ist die Schnittstelle eines Gerätes, die es mit

einem Netzwerk verbindet) dieses kann eine physische Schnittstelle (z.B. Ethernet-Anschluss, WLAN-Adapter) oder eine virtuelle(z.B.: VPN-Adapter) sein. Jedes Netzwerk-Interface hat eine MAC-Adresse und eine IP-Adresse vom Raspberry erweitert werden. Dafür muss die Konfigurationsdatei, die mit diesem Befehl geöffnet werden kann "sudo nano /etc/dhcp.conf", erweitert werden.

In diesem File muss das Interface für den Onboard-Ethernet-Anschluss bearbeitet werden.

```
interface eth0
static ip_address=192.168.1.100/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1
```

Abbildung 20: Netzwerk-Interface vom Raspberry Pi

Um die Bridge (Ein Netzwerk-Bridge ist eine Netzwerkschnittstelle, die mehrere physische oder virtuelle Netzwerkschnittstellen verbindet. Dabei fungiert eine Bridge ähnlich wie ein Switch, indem sie den ganzen Netzwerkverkehr weiterleitet) für die Ethernet-Adapter zu erstellen, wird ein Tool namens bridge-utils benötigt, dieses kann mit

```
sudo apt install bridge-utils -y
```

installiert werden. Normalerweise wird beim Installieren eines Pakets mit apt noch auf eine Bestätigung des Users gewartet. Mit dem "`-y`" wird diese sofort mit Ja beantwortet.

Um die Bridge zu erstellen und richtig zu konfigurieren, wird ein Skript erstellt.

```
sudo brctl addbr br0
sudo brctl addif br0 eth1
sudo brctl addif br0 eth2

# Bring up the interfaces
sudo ip link set dev eth1 up
sudo ip link set dev eth2 up
sudo ip link set dev br0 up
```

Abbildung 21: Unsere Bridge-Configuration

Mit dem Befehl "sudo brctl addbr br0" wird die Bridge br0 erstellt und mit "sudo brctl addif br0 eth1" und "sudo brctl addif br0 eth2" werden die zwei Ethernet-Adapter "eth1" und "eth2"

zur Bridge “br0“ hinzugefügt. Mit dem Befehl “sudo ip link set dev ____ up“ werden die drei Netzwerkschnittstellen “eth1“, “eth2“ und “br0“ aktiviert, so dass sie funktionsfähig sind.

Um automatisch ein neu angeschlossenes Ethernet-Adapter zum Bridge hinzuzufügen, wurde zuerst eine Udev-Regel erstellt. Udev ist der Geräte-Manager für den Linux-Kernel. Mit Udev-Regeln kann man definieren, wie bestimmte Geräte behandelt werden sollen, Berechtigungen für Geräte setzen und automatisch Skripte oder Programme ausführen, wenn ein Gerät erkannt wird. Udev-Regeln werden unter Raspberry Pi OS unter dem Pfad “etc/udev/rules.d/“ gespeichert.

Die Udev-Regel für das automatische Hinzufügen zur Bridge, wenn ein Ethernet-Adapter angeschlossen wird, sieht so aus:

```
1       ACTION=="add", SUBSYSTEM=="net", KERNEL=="eth*", RUN+="SCRIPTE/BRIDGE/add_to_bridge.sh
            %k"
```

Die Regel in dem Beispiel definiert mit “ACTION==“add““, dass diese Regel ausgeführt werden soll, wenn ein neues Gerät angeschlossen wird. Der Teil “SUBSYSTEM==“net“, KERNEL==“eth*““ besagt, dass diese Regel weithin nur für Netzwerkschnittstellen (“net“) des Typs Ethernet (“eth*“) ausgeführt werden soll. Treffen alle drei “Filter“ beim Hinzufügen eines Gerätes zu, wird das Skript “add_to_bridge.sh“, welches unter “SCRIPTE/BRIDGE/“ gespeichert ist, ausgeführt.

Diese wird dann wie schon erwähnt unter “etc/udev/rules.d/“ in einer .rules-Datei gespeichert. In diesem Fall würde diese Datei “99-usb.rules“ heißen.

Nach dem Hinzufügen von einer Udev-Regel muss das Udev-System neugestartet werden. Dafür gibt es zwei Befehle: “sudo udevadm control –reload-rules“ startet das Udev-System neu und “sudo udevadm trigger“ erzwingt eine Ausführung aller Udev-Regeln an den schon verbundenen Geräten. Da die Udev-Regel in diesem Projekt nicht funktioniert hat, wurde auf einen Workaround zurückgegriffen (Siehe §8.1.3 Skripte zur Erkennung und Einbindung von Geräten)

Autostart von Anwendungen

Damit das Backend des Projekts oder das Frontend auf dem Bildschirm des Raspberry Pis angezeigt wird, müssen diese automatisch gestartet werden. Dazu muss ein Skript erstellt werden, welches den Prozess startet.

```
1       chromium-browser --start-fullscreen --new-window https://10.130.145.54
```

Hier als Beispiel, wie der Browser im Vollbildmodus gestartet wird.

```
1       sudo dotnet /home/RaspiSniffer/rsyncTest/linkedFolder/linux-arm64/SniffingAPI.dll
```

Hier als weiteres Beispiel, wie das Backend gestartet wird.

Damit diese Skripte beim Start des Raspberry Pis automatisch ausgeführt werden, muss im Verzeichnis “~/.config/autostart“ eine .desktop-Datei erstellt werden. Eine Desktop-Datei ist eine Linux-Konfigurationsdatei, die definiert, wie eine Anwendung gestartet werden soll.

Diese .desktop-Dateien sehen wie folgt aus:

```
1       [Desktop Entry]
2       Version=1.0
3       Name=Chromium Autostart
4       Exec=/home/RaspiSniffer/Scripte/BROWSER/start_chromium_fullscreen.sh
5       Icon=Chromium
6       Type=Application
7       Terminal=false
```

Beim Erstellen einer .desktop-Datei müssen mindestens vier Einträge vorhanden sein. Jede .desktop-Datei startet immer mit “[Desktop Entry]“, alle darauffolgenden Abschnitte legen das Verhalten der Anwendung fest. Der Name definiert, wie diese Anwendung in Menüs benannt werden soll. Bei Exec steht der Befehl, der ausgeführt werden soll. In dem Beispiel ist der Pfad zu dem Bash-Skript zum Starten des Browsers im Vollbildmodus angegeben. Bei Type wird angegeben, was für eine Anwendung durch diese .desktop-Datei gestartet wird. Da Chromium ein Browser und daher eine Anwendung ist, steht im Beispiel “Application“. Der Aufbau einer .desktop-Datei ist standardisiert und sieht immer gleich aus.

Da diese .desktop-Dateien im Autostart-Ordner von Linux sind, werden sie bei jedem Start des Raspberry Pis auch ausgeführt.

Automatisches Mounten von USB-Sticks

In Raspberry Pi OS wird jeder USB-Stick mit udev und udisks2 automatisch erkannt und gemountet.

In Linux wird unter ‘mounten‘ der Vorgang gemeint, bei dem ein externes Dateisystem in das Dateisystem von Linux eingebunden wird, damit man darauf zugreifen kann.

Damit der mount Punkt verändert werden kann, muss die “mount_options.conf Datei“ von udisks2 bearbeitet werden. Standardmäßig ist dieser automatische mount Punkt der Pfad “/media/RaspiSniffer/“ Dabei steht RaspiSniffer für den Namen des angemeldeten Benutzers. Im Projekt wurde wegen der Einfachheit dieser Standardpfad nicht geändert.

Skripte zur Erkennung und Einbindung von Geräten

Da die Verwendung von Udev-Regeln zur automatischen Erkennung von einem neuen angeschlossenen Gerät bei diesem Projekt nicht funktioniert, wurde ein Skript erstellt. (Der Grund dafür wurde auch nicht vom Linux-Experten von Kontron gefunden.) Dieses Skript wird einmal beim Starten des Raspberry Pis durch eine .desktop-Datei (Für Erklärung von .desktop-Dateien siehe §8.1.3 Autostart von Anwendungen) ausgeführt. Damit es immer neue angebundene Ethernet-Adapter automatisch zur Bridge hinzufügt, läuft dieses Skript in einer Endlosschleife.

Listing 2: Check and Add to Bridge Skript

```
#!/bin/bash


while true; do
  output_file="eth_interface.txt"

  ls /sys/class/net | grep "eth" | grep -v "eth0" >"$output_file"

  while IFS= read -r eth_interface; do

    ./bridge.sh "$eth_interface"
  done < "$output_file"

  sleep 10
done
```

In diesem Skript wurde eine Datei definiert, wo alle angeschlossenen Ethernet-Adapter aufgelistet werden, um es fürs Backend einfacher zu machen, sich diese Liste zu holen, um ans Frontend weiterzugeben. In Linux werden alle angeschlossenen Netzwerkgeräte im Pfad "/sys/class/net/" aufgelistet. Dieses Skript holt sich alle Interfaces, außer eth0, welches der Ethernet-Anschluss am Raspberry Pi ist, und schreibt den Namen von diesen in die Ausgabedatei. eth0 wird ignoriert, weil es für das Hosten des Frontends zuständig ist und nicht durch netsniff-ng abgehört werden soll. Für alle Ethernet-Anschlüsse außer eth0 wird dann das Skript "bridge.sh" ausgeführt.

Nach 10 Sekunden wird dieses Skript nochmals ausgeführt.

Listing 3: Bridge.sh Skript

```
#!/bin/bash

BRIDGE_IFACE=br0
LOGFILE="/tmp/add_to_bridge.log"

echo "Script started" >> $LOGFILE

display_device_info_and_add_to_bridge() {
    local iface_name="$1"
    local device_path="/sys/class/net/$iface_name"

    echo "Device Path: $device_path" >> $LOGFILE
    echo "Interface Name: $iface_name" >> $LOGFILE

    local devtype=$(udevadm info --query=property --path="/class/net/$iface_name" 2>>
        $LOGFILE | grep ID_USB_DRIVER | cut -d= -f2)
    echo "Device Type fetched: $devtype" >> $LOGFILE

```

```
18      if [[ "$devtype" == *"r8152"* || "$devtype" == *"cdc_ether"* || "$devtype" ==
            *"ax88179_178a"* ]]; then
19          echo "This is a USB to Ethernet adapter. Adding to bridge $BRIDGE_IFACE." >>
                $LOGFILE
20
21          sudo ip link set dev $iface_name up >> $LOGFILE 2>&1
22          if [[ $? -ne 0 ]]; then
23              echo "Failed to bring interface $iface_name up" >> $LOGFILE
24          else
25              echo "Interface $iface_name brought up" >> $LOGFILE
26          fi
27
28          sudo brctl addif $BRIDGE_IFACE $iface_name >> $LOGFILE 2>&1
29          if [[ $? -ne 0 ]]; then
30              echo "Failed to add interface $iface_name to bridge $BRIDGE_IFACE" >> $LOGFILE
31          else
32              echo "Interface $iface_name added to bridge $BRIDGE_IFACE" >> $LOGFILE
33          fi
34      else
35          echo "Device is not a recognized USB to Ethernet adapter" >> $LOGFILE
36      fi
37  }
38
39  iface_name="$1"
40  echo "Received interface name: $iface_name" >> $LOGFILE
41
42  if [[ -d "/sys/class/net/$iface_name" ]]; then
43      echo "Interface $iface_name exists, proceeding with device info fetch" >> $LOGFILE
44      display_device_info_and_add_to_bridge "$iface_name"
45  else
46      echo "Interface $iface_name does not exist" >> $LOGFILE
47  fi
48
49  echo "Script ended" >> $LOGFILE
```

Dieses Skript kriegt von dem vorherigen Skript (Check and Add to Bridge Skript) den Namen des Interfaces (Beispiel: eth2). Bei diesem wird überprüft, ob es auch wirklich ein Ethernet-Anschluss ist und fügt diesen dann zu der Bridge br0 hinzu (für die Erstellung von br0 siehe §8.1.3 Raspberry Pi als Bridge einrichten). Damit man in einem Fehlerfall einfach herausfinden kann, was den Fehler ausgelöst hat, wird alles in einer Log-Datei (Protokolldatei) geschrieben. Diese Log-Datei wird im "/tmp/"-Pfad gespeichert.

Erstellung von WiFi-Verbindungsskripten

Damit man mit einem Bash-Skript eine Verbindung zu einem WLAN herstellen kann, braucht man nmcli. Das ist eine Software, die es ermöglicht, mit einfachen Befehlen WLAN-Verbindungen herzustellen. Diese wurde mit dem Befehl

```
1       nmcli connect device
```

Dafür gibt es drei verschiedene Skripte im Raspberry Pi. "Connect_to_Wifi.sh", "Connect_to_hidden_Wifi.
und "Disconnect_from_Wifi.sh".

Listing 4: Wlan Verbindungs Skript

```
1   #!/bin/bash
2
3   get_security_type() {
4       local ssid=$1
5       nmcli -t -f ssid,security device wifi list | grep "^${ssid}:" | cut -d ':' -f2
```

```
}
if [ "$#" -lt 2 ]; then
    echo "Usage: $0 [SSID] [PASSWORD] <USERNAME>"
    exit 1
fi

SSID=$1
PASSWORD=$2
USERNAME=$3

start_time=$(date +%s)

if ! command -v nmcli &> /dev/null; then
    echo "Error: nmcli is not installed. Please install it and try again."
    exit 1
fi

SECURITY=$(get_security_type "$SSID")

echo "$SSID: $SECURITY"

case $SECURITY in
    "--" | "" | " ")
        echo "Connecting to open network: $SSID"
        nmcli device wifi connect "$SSID"
        ;;
    "WEP")
        echo "Connecting to WEP Network: $SSID"
        nmcli device wifi connect "$SSID" password "$PASSWORD"
        ;;
    *"WPA"*)
        if [[ "$SECURITY" == *"802.1X"* ]]; then
            if [ -z "$USERNAME" ]; then
                echo "Error: Username required for $SECURITY network"
                exit 1
            fi
            echo "Connecting to WPA2/WPA3-Enterprise network: $SSID"

            # Remove existing connection profile if it exists
            nmcli connection delete "$SSID" &> /dev/null

            # Create a new connection profile for WPA2/WPA3-Enterprise
            nmcli connection add type wifi ifname wlan0 con-name "$SSID" ssid "$SSID" \
                wifi-sec.key-mgmt wpa-eap \
                802-1x.identity "$USERNAME" \
                802-1x.password "$PASSWORD" \
                802-1x.eap peap \
                802-1x.phase2-auth mschapv2 \
                wifi-sec.psk "$PASSWORD"

            nmcli connection up "$SSID"
        else
            echo "Connecting to WPA/WPA2/WPA3 network: $SSID"
            nmcli device wifi connect "$SSID" password "$PASSWORD"
        fi
        ;;
    *)
        echo "Unsupported or unknown security type: $SECURITY"
        exit 1
        ;;
esac

for i in {1..10}; do
    nmcli connection show --active | grep "$SSID" &> /dev/null
    if [ $? -eq 0 ]; then
        end_time=$(date +%s)
        elapsed=$((end_time - start_time))
        echo "Successfully connected to $SSID in $elapsed seconds."
        exit 0
    fi
    sleep 1
done

echo "Failed to connect to $SSID."
```

Im Frontend werden die Parameter wie SSID (Der Name eines WLANs), Benutzername und Passwort eingegeben und an das Backend weitergegeben. Das Backend startet dann dieses Skript mit den übergebenen Parametern. Am Anfang des Skripts wird zuerst die Sicherheit des WLANs

überprüft. Bei Wlans gibt es mehrere Sicherheitsstufen, diese sind WPA, WPA2, WPA3, WEP, offen und 802.1X. Abhängig von der Sicherheitsstufe wird auch ein Benutzername benötigt oder nicht. Das Skript geht alle Sicherheitsstufen durch und wendet dann alle nötigen Parameter an, um eine Verbindung aufzubauen.
Am Ende wartet das Skript 10 Sekunden und misst dabei die Zeit. Wurde erfolgreich eine Verbindung hergestellt, gibt es eine Erfolgsnachricht und die benötigte Zeit ans Backend weiter, damit dieses diese ans Frontend weiterleiten und anzeigen kann.

Für versteckte WLANs sieht die Verbindung leicht anders aus. Um zu einem normalen WLAN zu verbinden, muss man bei nmcli nur Folgendes angeben:

```
nmcli device wifi connect "$SSID"
```

Bei einem versteckten Wlan sieht der nmcli-Befehl so aus:

```
nmcli device wifi connect "$SSID" hiddden yes
```

Das dritte und letzte Skript ist rein dafür da, um eine bestehende Wlan-Verbindung zu trennen.

Listing 5: Trennen einer Wlan Verbindung

```
#!/bin/bash

get_active_wifi() {
  nmcli -t -f TYPE,STATE,CONNECTION device | grep '^wifi:connected:' | cut -d ':' -f3
}

ACTIVE_WIFI=$(get_active_wifi)

if [ -z "$ACTIVE_WIFI" ]; then
  echo "No active WIFI connection found"
  exit 1
fi

echo "Disconnecting from WIFI network: $ACTIVE_WIFI"

nmcli connection down "$ACTIVE_WIFI"

if [ $? -eq 0 ]; then
  echo "Successfully disconnected from $ACTIVE_WIFI"
else
  echo "Failed to disconnect form $ACTIVE_WIFI"
fi
```

Um die Verbindung zu trennen, wird die SSID der gerade aktiven WLAN-Verbindung geholt. Dies geschieht durch “nmlci -t -f TYPE,STATE,CONNECTION device | grep ’ŵifi:connected:’ | cut -d ’:’ -f3“. Der erste Teil dieser Abfrage holt sich nur den Typ der Schnittstelle (WLAN oder Ethernet), den Status (connected oder disconnected) und falls der Status “connected“ ist, wird noch die gerade aktive Verbindung angezeigt. Der Teil “grep ’ŵifi:connected:’“ sorgt dafür, dass nur der Eintrag, wo eine aktive WLAN-Verbindung besteht, angezeigt wird. Der letzte Teil “cut -d ’:’ -f3“ trennt die Ausgabe “wifi:connected:WLAN_Home“ (nur ein Beispiel, wie es aussehen könnte) bei den Doppelpunkten und speichert den dritten Teil. In der Beispielausgabe wäre das

“WLAN_Home“ was die SSID ist. Wenn es keine aktive WLAN-Verbindung gibt, wird ein leerer Text gespeichert.

Da “nmcli -t -f TYPE,STATE,CONNECTION device | grep ’ŵifi:connected:’ | cut -d ’:’ -f3“ einen Text speichert und wenn keine aktive WLAN-Verbindung vorhanden ist, dieser leer ist, muss überprüft werden, welcher Fall eingetreten ist. Falls der Text leer ist, wird das Skript abgebrochen und dem Backend gesagt, dass gerade keine aktive WLAN-Verbindung vorhanden ist. Falls aber im Text etwas steht, wird die Verbindung mit “nmcli connection down "$ACTIVE_WIFI"getrennt.

Der letzte Teil des Skripts sagt dem Backend, ob die Trennung erfolgreich oder fehlgeschlagen ist.

8.1.4 Webapplikation

Einrichtung des Backends und Frontends auf dem Raspberry Pi

Das Backend und Frontend wurden auf PCs mit Windows entwickelt. Die zwei Projekte wurden dann für die arm-64 CPU-Architektur veröffentlicht und mit SSH auf den Raspberry Pi kopiert. SSH ist die Abkürzung für Secure Shell. Dieses ermöglicht es, eine verschlüsselte Verbindung zwischen zwei Geräten zu erstellen. Dabei kann man das andere Gerät über die Konsole steuern. Es wird für die Fernadministrierung von Servern oder einfachen Dateitransfer zwischen zwei Geräten verwendet.

Das Backend konnte dann durch ein einfaches Skript gestartet werden, welches auch automatisch beim Starten des Raspberry Pi ausgeführt wird.

Das Frontend hingegen wurde von Nginx gehostet. (Siehe §8.1.4 Nutzung von Nginx für Hosting)

Nutzung von Nginx für Hosting

Damit man Nginx überhaupt verwenden kann, muss man es zuerst mit

```
sudo apt install nginx -y
```

installieren. Danach muss man eine neue Nginx-Konfigurationsdatei erstellen.

```
server {
    listen 80;
    server_name "IP-Adresse vom Raspberry Pi";

    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
```

```
}

server {
    listen 443 ssl;
    server_name "IP-Adresse vom Raspberry Pi";

    root /home/pi/blazor-app;
    index index.html;

    ssl_certificate /etc/ssl/private/company_cert.crt;
    ssl_certificate_key /etc/ssl/private/company_key.key;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /_framework/ {
        gzip_static on;
        expires max;
        add_header Cache-Control "public, max-age=31536000, immutable";
    }
}
```

In dieser Konfigurationsdatei werden der Port und die IP-Adresse des Webservers definiert. Der erste Abschnitt definiert, dass jede HTTP-Anfrage auf den HTTPS-Block weitergeleitet werden soll. Im zweiten "server"-Abschnitt wird der Pfad (in diesem Fall "/etc/ssl/private") für die Zertifikate angegeben, diese Zertifikate sind nötig, wenn das Frontend über HTTPS gehostet wird (Grund in §8.1.4 Fehlerbehebung bei der Weboberfläche genauer erklärt). Zusätzlich wird hier angegeben, wo nginx die Blazor WebAssembly-Dateien, die mit SSH kopiert wurden, findet. Der "location" und "location /_framework/"-Abschnitt definiert nötige Optimierungen für das Hosten von einer Blazor WebAssembly Single-Page-Application. [26][26]

Fehlerbehebung bei der Weboberfläche

Anfangs gab es nur HTTP und man konnte nicht mit jedem Browser auf die vom Raspberry Pi gehostete Website zugreifen, weil HTTP ein unsicherer Weg ist, zu kommunizieren. Wird in allen modernen Browsern eine Warnung angezeigt, wenn man darauf zugreifen wollte. Um keine solche Warnung zu erhalten, muss die Website über HTTPS gehostet werden. Dafür ist aber ein signiertes Zertifikat nötig. So ein Zertifikat kann man bei einer Zertifizierungsstelle wie "Let's Encrypt" beantragen. Für dieses Projekt wurde von Kontron ein gültiges Zertifikat bereitgestellt. Wie dieses eingebunden wurde, ist in §8.1.4 Nutzung von Nginx für Hosting erklärt. Nach dem Einbinden des genannten Zertifikates war es möglich, über HTTPS mit jedem Browser auf die Website zuzugreifen.

[26] *https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-9.0*

8.1.5 Fehlerbehebung und Workarounds

Probleme mit Udev-Regeln und alternative Ansätze

Wie im Abschnitt §8.1.3 Skripte zur Erkennung und Einbindung von Geräten schon erwähnt wurden, wurde aus einem nicht erklärlichen Grund die Udev-Regel nicht ausgeführt. Deswegen wurde der Workaround einer Endlosschleife verwendet.

Lösungen für Autostart und Performance-Optimierung

Bei dem automatischen Starten des Browsers gab es anfangs das Problem, dass der Browser nicht im Vordergrund oder "unsichtbar" gestartet ist. Dieser Fehler wurde einfach behoben, indem wir statt

```
chromium_browser --start-fullscreen --app
```

```
chromium_browser --start-fullscreen --new-window
```

in dem "start_chromium_fullscreen.sh" Skript verwendet haben

8.1.6 Testplan und Validierung

Testberichte und Anpassungen

Es wurde ein Testplan erstellt, um alle Funktionen vom RaspiSniffer zu testen. Beim Durchlaufen des Testplans wurde festgestellt, dass man nur WLAN oder Ethernet sniffen kann.
Dieses Problem wurde einfach gelöst, indem im Backend der 'netsniff-ng' Befehl mit weiteren Parametern aufgerufen wurde. Dies ermöglichte diese vier Sniffing-Möglichkeiten:

- WLAN + Ethernet
- Nur WLAN
- Die Bridge br0
- Nur ein Ethernet

Herausforderungen bei den Tests

Es gab einige Probleme, die Tests zu erstellen, denn wir hatten damit noch keine Erfahrung. Aber unser Betreuer bei Kontron hat gemeint, wir hätten diese Hürde erfolgreich gemeistert.

8.2 Backend

In dieser Diplomarbeit ist das Backend für die Bereitstellung aller wichtigen Daten und das Ausführen der Bash-Skripte auf dem Raspberry Pi verantwortlich. Es speichert die wichtigsten Daten der letzten Pakete, die vom RaspiSniffer abgefangen wurden. Außerdem bietet es verschiedene Funktionen zur Kontrolle des Raspberry Pi.

8.2.1 Erstellung der API-Solution

Um das Backend zu erstellen, haben wir Rider verwendet. Dieses Programm erleichtert die Konfiguration der API. Wir haben uns für diesen Ansatz entschieden, weil Rider automatisch alle benötigten Dateien erstellt. Die für uns richtige Solution war die Web API unter dem Project-Type Web 22.

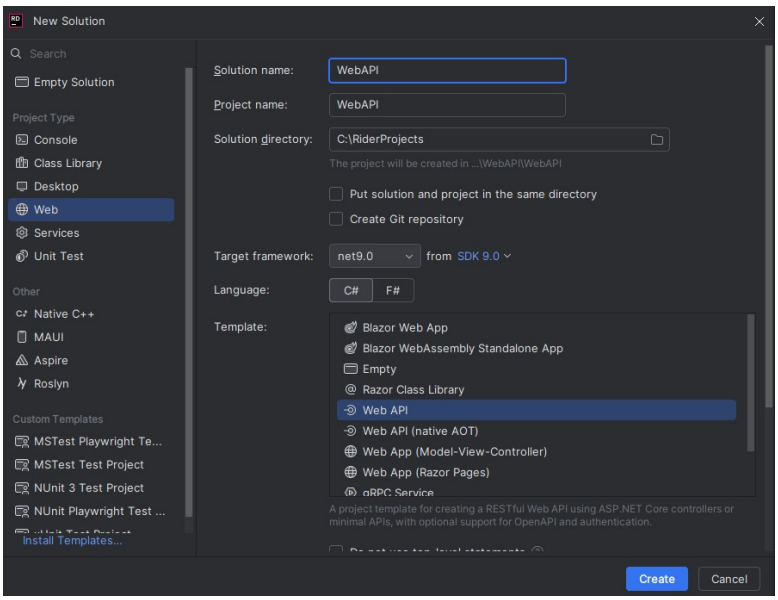

Abbildung 22: Solution erstellen mit Rider

8.2.2 Modell Klassen

Die Modellklassen sind Klassen, deren einzige Funktion das Speichern von Daten ist. Sie werden verwendet, um Datenspeicherung und Datentransfers einfacher zu machen.

Wifi-Klasse

Die Wifi-Klasse speichert wichtige Details zu jedem WLAN-Netzwerk. Diese Klasse wird verwendet, um gescannte WLAN-Netzwerke zu speichern. Die wichtigsten Attribute der Klasse Wifi sind „Ssid“, „IsConnected“ und „SecurityType“. Das Attribut Ssid speichert die SSID des WLAN-Netzwerks, also den Namen des Netzwerks. IsConnected speichert einen booleschen Wert, der darstellt ob der Raspberry Pi in dem Moment mit diesem Netzwerk verbunden ist. Das Attribut SecurityType speichert die Authentifizierungsmethode, die dieses Netzwerk benutzt, um User zu identifizieren. Meistens ist es entweder open, WPA2 oder WPA2-Enterprise. Dieses Attribut ist wichtig, damit sich der Raspberry Pi mit dem WLAN-Netzwerk verbinden kann, da verschiedene Security Types User verschieden authentifizieren. Falls der RaspiSniffer versucht, sich mit einem bestimmten WLAN-Netzwerk zu verbinden, wird das SecurityType-Attribut an das Frontend geschickt, damit der Benutzer die richtigen Daten eingeben kann.

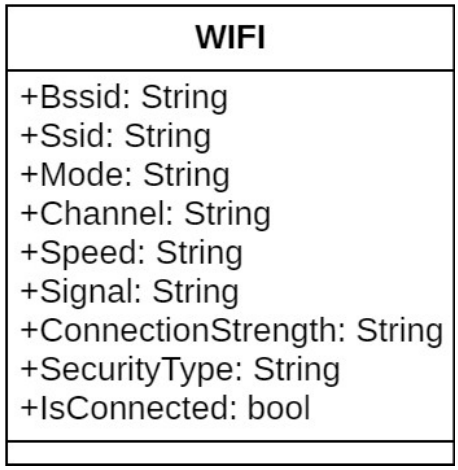

Abbildung 23: UML Diagramm der Wifi-Klasse

Packet-Klasse

Die Packet-Klasse speichert wichtige Informationen über die Pakete die der Raspberry Pi abhört. Das einzige Attribut, das nicht im Frontend angezeigt und daher nur kaum verwendet wird, ist die Liste TerminalOutputLines. Diese speichert die genaue Ausgabe des Befehls, der in der Linux-Shell ausgeführt wurde, um das Packet-Objekt zu erstellen. Somit speichert es unter anderem die Daten, die in einem Packet enthalten sind. Da diese jedoch meistens verschlüsselt sind, lesen sich diese Daten wie zufällig generierte Buchstaben. Diese Klasse hat kurzzeitig ein großes Problem in dem Code des Backend offenbart. Die API hat in den ersten Wochen eine unbeschränkte Anzahl an Packet-Objekten gespeichert. Wenn der RaspiSniffer mit diesen Einstellungen eine aktive Verbindung überwacht hat, war der Raspberry Pi schnell überfordert.

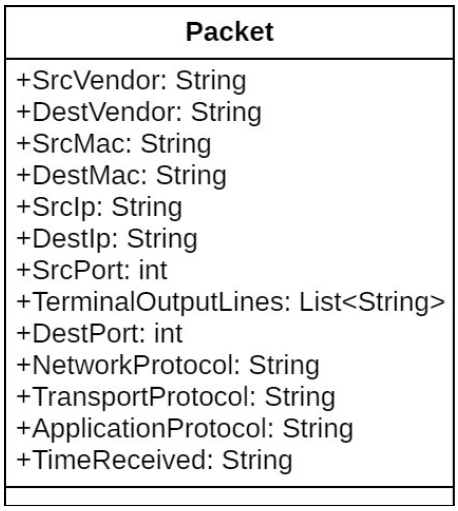

Abbildung 24: UML Diagramm der Packet-Klasse

8.2.3 Datenspeicherung

Um die Daten im Backend effizient zu speichern, wurden mehrere Klassen verwendet, deren primäre Funktion es ist, die Daten des Backends zu speichern.

Datastore

Der Datastore ist eine Klasse, die verschiedene wichtige Daten des Programms speichert. Sie wurde mit dem Design Pattern Singleton entworfen, damit alle Threads der ASP.NET Core API auf den gleichen Datenbestand zugreifen können. Die wichtigsten Attribute, die diese Klasse speichert, sind die zwei CancellationTokenSource Objekte. Diese Objekte ermöglichen es, einen Befehl, der mit der CLI.wrap Bibliothek gestartet wurde, wieder zu beenden. Sie werden für den Kopiervorgang und den netsniff-ng Befehl verwendet. Diese Attribute sind notwendig, damit ein User sie unterbrechen kann.

Um die Benutzung zu vereinfachen, erstellt der Datastore eine neue CancellationTokenSource Instanz, wenn die ältere bereits aktiviert wurde.

Circular Buffer

Der CircularBuffer ist eine Klasse, die eine „circular linked list25“ imitieren soll. Diese Architektur besteht aus beliebig vielen Objekten, die jeweils auf das nächste in der Reihe verweisen, wobei das letzte Objekt auf das erste verweist. Dadurch wird die linked list „circular“, auf Deutsch ringförmig. Die Anordnung einer solchen Linked List bedeutet, dass an einem bestimmten Punkt alte Daten mit neuen überschrieben werden. Der CircularBuffer verwendet keine Linked List. Um die Verwaltung einfach zu gestalten, verwendet sie ein Array mit bestimmt vielen Einträgen. Die Klasse emuliert stattdessen den Effekt der „Ringförigen Linked List“ mit einem _start Wert und einem _end Wert. Der _start Wert ist der Index des ältesten Eintrags. Dieser Eintrag wird überschrieben, wenn ein neuer Wert gespeichert wird. Der _end Wert speichert den neuesten Eintrag. Dieser Wert wird benötigt, damit die Klasse ein richtig formatiertes Array, also ein Array, in dem der jüngste Eintrag an erster Stelle und der älteste Eintrag an letzter Stelle ist, erstellen kann. Diese Klasse wurde erstellt, um den RAM-Verbrauch des

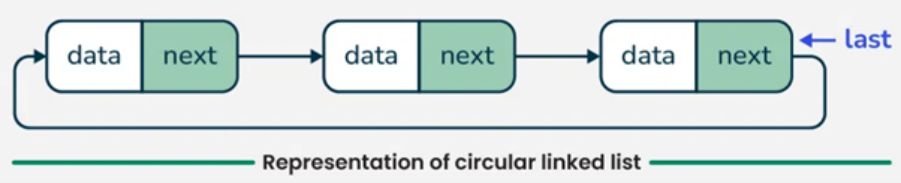

Abbildung 25: Darstellung einer Circular Linked List

Backend drastisch zu reduzieren. Bevor diese Klasse einfegührt wurde, konnte der RaspiSniffer nur für kurze Zeit laufen, bevor er keinen freien RAM mehr zur Verfügung hatte. Die gesnifften Pakte, wurden gespeichert aber nie gelöscht. Diese Klasse löste das Problem, indem sie nur 500 Pakete gleichzeitig speichert. Wir haben diese Zahl gewählt, damit das Frontend alle relevanten Daten bekommt und das Backend trotzdem nur wenig RAM benötigt.

8.2.4 Program

Die Program.cs Datei ist das Herzstück der API. In dieser Klasse wird die API konfiguriert, es werden die wichtigsten Skripts gestartet, und die Controller werden an das Projekt angefügt. Sie wird als erstes ausgeführt, wenn das Backend neu gestartet wird. Obwohl diese Klasse für das Backend sehr wichtig ist, ist sie sehr kurz. Die Variable builder, bietet die meisten Konfigurationsmöglichkeiten.

Der erste Schritt, den fast jede ASP.NET Core Applikation durchführt, ist es, so einen builder anzulegen. Die Methode „WebApplication.CreateBuilder(args)" gibt einen builder zurück, der zum Teil konfiguriert wurde. Außerdem erlaubt diese Methode es Argumente zu übergeben. Meistens werden die Befehlszeilenargumente der Program.cs Datei übergeben, damit der Nutzer des Programms diese Konfiguration verändern kann, ohne den Code zu verändern.

Der nächste Befehl ist für diese Diplomarbeit spezifisch. Hier versucht das Programm, durch die Datastore Klasse, die IP-Adresse des eth0 Anschlusses herauszufinden. Das Ziel dieser Operation ist es, die API nur auf diesem Ethernet-Port zur Verfügung zu stellen, damit Anfragen an die API nicht vom RaspiSniffer als abgefangene Kommunikation zählen. Falls es bei diesem Befehl zu einem Fehler kommt, verwendet die API die IP-Adresse 0.0.0.0 als Ersatz. Mit dieser Einstellung kann die API über alle Ethernet Anschlüsse erreicht werden. Um diese Settings wirklich im builder zu verändern, wird die „builder.WebHost.UseUrls" Methode verwendet.

Damit das Frontend auf allen Computern in jedem möglichen privaten Netzwerk die Daten des Backend abfragen kann, muss eine CORS-Policy im Program.cs definiert werden. Diese Policy muss Anfragen von jedem beliebigen Host erlauben, da die Daten, die die API bereitstellt, nicht geschützt sind und für die Öffentlichkeit verfügbar sein sollen. Damit die Policy von der Web API wirklich verwendet wird, muss sie am Ende der Konfiguration, aber noch vor dem Starten, in der „UseCors()" Methode definiert werden.

Um die ASP.NET Core API von einer einzigen Datei zu einem MVC-Projekt zu erweitern, müssen die Controller in das Projekt hinzugefügt werden. Dafür wird die Methode „builder.Services.AddControllers()" benötigt. Diese Methode sucht nach allen öffentlichen Klassen,

die von der Klasse Controller oder von der Klasse ControlerBase erben, und fügt die Endpunkte, die diese Klasse bereitstellt, an die API an[27][27]. Außerdem besitzt ein Controller meistens eine „[ApiController]“ Annotation und eine „[Route()]“ Annotation, die den Pfad angibt, an dem der Controller durch die API erreichbar ist. Wenn eine Anfrage an einen Endpunkt geschickt wird, der von einem Controller bereitgestellt wird, wird eine neue Instanz des Controllers angelegt, um die Anfrage zu bearbeiten.

Nachdem alle Einstellungen an dem builder Objekt vorgenommen wurden, wird mit der Methode „builder.Build()“ ein app Objekt erstellt. Dieses Objekt beinhaltet ein fertiges WebApplication Objekt, das weniger Einstellungsmöglichkeiten bietet als das builder Objekt.

Am Ende der Program.cs Datei, wenn alle Konfigurationen eingestellt wurden, wird die API gestartet.

8.2.5 Controller

Packet Controller

Die PacketController Klasse implementiert alle Endpunkte der API. Sie speichert verschiedene Command Objekte aus der Bibliothek CLI.Wrap, Instanzen von allen Model Klassen und andere Daten, die in der Bereitstellung der Daten benötigt werden. Manche dieser Objekte sind als statische Objekte deklariert, damit sie ihre gespeicherten Daten nicht verlieren, wenn die Controller Instanz gelöscht wird. Die CircularBuffer Instanz, zum Beispiel, wird in mehreren Instanzen benötigt. Wenn dieses Objekt nicht als statisches Objekt gespeichert wäre, könnte ein Controller, der erstellt wird, nicht auf Pakete zugreifen, die andere Instanzen des PacketController gespeichert haben.

8.2.6 Endpoints

Pakete

Get /api/packets/ Dieser Endpoint wird verwendet, um einem Client die aktuellsten Pakete zu schicken. Diese Pakete werden in einem Ringspeicher gespeichert. Dieser Speicher ist ein statisches Objekt. Dadurch speichert das Backend nur ein begrenzt großes Array an Daten und schickt daher maximal 500 Pakete auf einmal an das Frontend. Falls der Raspberry Pi den Ringspeicher noch nicht völlig aufgefüllt hat, schickt das Backend ein kleineres Array zurück.

[27] *https://learn.microsoft.com/de-de/aspnet/core/mvc/controllers/actions?view=aspnetcore-9.0*

Input: Dieser Endpoint benötigt keine Input-Daten.

Output: Der Output ist eine Liste an Packet-Objekten. Diese werden im JSON-Format verschickt. Die Liste hat eine begrenzte Größe, die nie überschritten werden kann. In dieser Diplomarbeit ist die Größe auf 500 Pakete beschränkt, da eine höhere Anzahl den RAM des Raspberry Pi zu stark belastet hätte. Die Anzahl kann einfach verändert werden, indem man eine andere Zahl bei der Initialisierung des Ringspeichers verwendet.

Umsetzung: Da das Backend laufend Pakete speichert, während der Sniffing-Befehl läuft, muss diese Funktion nur die bereits gespeicherten Daten in das JSON-Format umwandeln und den daraus resultierenden String verschicken.

Listing 6: Endpoint im Code

```
1       [HttpGet("/api/packets/")]
2       public IActionResult GetAll()
3       {
4           return Ok(JsonSerializer.Serialize(Packets.ToArray())); // returns ring buffer as
                array
5       }
```

Aufname

Post /api/startRec/ Diese Methode wird verwendet, um den Sniffing-Befehl zu starten. Außerdem kann ein Benutzer mit dieser Methode einen Sniffing-Vorgang planen. Dieser startet zu einer bestimmten Zeit, und kann entweder laufen, bis er manuell beendet wird, oder bis zu einem anderen beliebigen Zeitpunkt.

Input: Der Input für diesen Endpoint besteht aus einer Liste an Strings die zwischen zwei und fünf Werte haben kann. Diese sind:

- Die erste Option muss verwendet werden. Sie ist der Filter, der von netsniff-ng verwendet werden soll. In diesem Filter können bestimmte Protokolle oder Ports definiert werden, die aufgezeichnet werden sollen. Das Programm verwendet genau den Filter, der übergeben wird. Also hat ein User alle Optionen, die netsniff-ng zur Verfügung stellt.
- Die zweite Option ist auch obligatorisch. Sie gibt an, welche Netzwerke der RaspiSniffer abhören soll. Es kann entweder das WLAN-Netzwerk oder alle angeschlossenen Ethernet-Adapter abgehört werden.

- Die dritte Option ist die erste, die nicht definiert werden muss. Sie bestimmt den Zeitpunkt, an dem der Sniffing-Vorgang starten soll. Falls kein Zeitpunkt ausgewählt wird, startet das Backend den netsniff-ng Befehl sofort nachdem die Anfrage eintrifft.
- Die vierte Option ist auch nicht obligatorisch. Diese gibt den Zeitpunkt an, an dem der Sniffing-Vorgang gestoppt werden soll. Falls diese Option nicht ausgewählt wird, läuft der Sniffing-Befehl, bis er manuell gestoppt wird.
- Die fünfte und letzte Option wurde in dieser Diplomarbeit nicht genützt. Sie definiert den Namen, den die Datei bekommen soll, die von netsniff-ng erstellt wird. Da diese Funktion keine wichtige Anforderung der Diplomarbeit war, wurde sie aus Zeitgründen entfernt. Die Option besteht aber weiterhin im Backend.

Output: Der Endpoint gibt keine Daten an das Frontend zurück. Stattdessen kann entweder ein „No Contentt“Status-Code oder ein „Bad Request“Status-Code entstehen.

Umsetzung: Das Programm beginnt damit, die Start- und Endzeitpunkte, falls vorhanden, in das DateTime Format zu konvertieren. Diese Zeitpunkte werden danach in einer statischen Variable gespeichert, damit sie in anderen Methoden zur Verfügung stehen. Als nächstes erstellt das Backend eine Variable, die den Dateinamen der von netsniff-ng erstellten Datei speichert. Damit das Programm den Datenverkehr eines WLAN-Netzwerks und den eines Ethernet Netzwerks gleichzeitig abhören kann, besitzt es zwei Befehle, die unabhängig voneinander verändert oder gestartet werden können. Deshalb fügt das Programm den vom Nutzer definierten Filter und den generierten Dateinamen beiden netsniff-ng Befehlen hinzu. Nur wenn kein Filter oder nur ein bestimmter Port definiert ist, wird dieser Schritt leicht verändert. Falls der User keine Filter angegeben hat, wird nur der Dateiname verändert. Wenn der Filter nur einen bestimmten Port enthält, muss der netsniff-ng Befehl etwas anders konfiguriert werden, um Fehler zu vermeiden.

Der letzte Schritt, den das Backend ausführt, ist der Start der Befehle. Falls der Nutzer keine Endzeit eingestellt hat, wird der Befehl ohne Veränderung ausgeführt. Falls ein Endzeitpunkt definiert wurde, fügt das Programm dem Befehl einen weiteren CancellationToken hinzu. Dieser Token wird automatisch aktiviert, wenn der Endzeitpunkt eintrifft. Da CLI.Wrap nur einen CancellationToken verwenden kann, benutzt das Backend die Methode CancellationTokenSource.CreateLinkedTokenSource(). Mit dieser Methode können mehrere CancellationTokens zusammengesetzt werden, sodass der Vorgang abgebrochen wird, wenn ein Token aktiviert wird.

Get /api/schedule Dieser Endpoint stellt Informationen über einen geplanten Sniffing-Vorgang zur Verfügung. Damit das Frontend Informationen über einen geplanten Sniffing-Vorgang anzeigen kann, muss es auch im Backend einen Endpoint geben, der diese Daten zur Verfügung stellt. Im Backend werden die Startzeit und Endzeit eines geplanten Sniffs als statische Variablen gespeichert, damit selbst wenn ein neuer Thread für eine neue Anfrage gestartet wird, dieselben Daten gespeichert sind. Außerdem garantiert dieser Ansatz, dass nur ein Sniff auf einmal geplant sein kann. Diese gespeicherten Daten werden auf Anfrage an das Frontend übergeben, damit User eine Bestätigung haben, dass die Planung erfolgreich war und wann der Befehl gestartet wird.

Input: Dieser Endpoint benötigt keine Inputs.

Output: Die Ausgabe des Endpoints ist ein JSON-String, der zwei Strings in der Form eines Arrays speichert. Falls kein Ende für einen Sniffing-Vorgang geplant ist, ist der zweite Eintrag des Arrays leer. Falls kein Sniff gestartet wurde, sind beide Einträge leer.

Umsetzung: Das Programm überprüft die Start- und Endzeit des geplanten Sniffing-Vorgangs. Falls eine Zeit festgelegt wurde, wird diese in ein lesbares Format geschrieben und in einem Array gespeichert. Nachdem beide Zeitpunkte in das passende Format umgewandelt wurden, wird das Array in einen JSON-String geschrieben und zurückgeschickt.

Get /api/isRecording/ Das Frontend muss herausfinden können, ob der RaspiSniffer zu einem bestimmten Moment Pakete aufzeichnet. Diese Funktion ist wichtig, damit Nutzer nicht versuchen, mehrere Sniffing-Vorgänge auf einmal zu starten. Stattdessen sollte das Frontend eine Option anbieten, mit welcher der Sniffing-Vorgang gestoppt werden kann.

Input: Dieser Endpoint benötigt keinen Input.

Output: Der Endpoint gibt einen String zurück, der entweder „true“oder „false“ist.

Umsetzung: Um herauszufinden, ob ein Sniffing-Vorgang läuft, verwendet das Backend den dazugehörigen CancellationToken. Falls dieser nicht unterbrochen wurde, läuft mindestens ein Sniffing-Befehl. Sobald der Token aktiviert wird, bricht der Befehl ab.

Get /api/stopRec/ Der letzte wichtige Endpoint, der mit der Aufzeichnung der Pakete direkt interagiert, wird benutzt, um alle netsniff-ng Befehle zu stoppen. Nachdem diese gestoppt wurden, werden die Aufnahmedateien erstellt.

Input: Dieser Endpoint benötigt keine Input-Daten.

Output: Der Endpoint schickt nur einen Status-Code zurück. Entweder einen „No Contentt“Status-Code oder im Falle eines Fehlers einen „Bad Request“Status-Code.

Umsetzung: Die Methode beginnt damit, den CancellationToken, der in allen Sniffing-Befehlen verwendet wird, zu aktivieren. Dadurch werden alle laufenden Befehle gestoppt. Außerdem setzt das Programm die Zeitpunkte, die für einen geplanten Sniffing-Vorgang verwendet werden, wieder auf null.

Dateien

Get /api/recordings/ Ein wichtiger Teil des RaspiSniffers ist die Funktion, Dateien herunterladen zu können. Diese Dateien sind die bisher durchgeführten Aufnahmen. Diese werden in Dateien mit der Endung .pcap gespeichert, damit sie von Analyseprogrammen wie Wireshark verwendet werden können. Damit ein User auswählen kann, welche Dateien er herunterladen will, gibt es diesen Endpoint, der alle .pcap Dateien, die zum Download stehen, zurückgibt.

Input: Dieser Endpoint benötigt keinen Input.

Output: Der Rückgabewert, den dieser Endpoint produziert, ist eine Liste an Strings. Jeder Eintrag in dieser Liste ist der Dateiname einer Aufnahmedatei, die im RaspiSniffer gespeichert ist.

Umsetzung: Das Programm führt einen Befehl aus, der alle Dateien mit der Dateiendung .pcap, die in einem bestimmten Ordner gespeichert sind, auflistet. Dieser Ordner wird verwendet, um alle Dateien, die von netsniff-ng produziert wurden, zu speichern. Nachdem der Befehl ausgeführt wurde, schickt das Backend ein String Array mit allen Dateinamen zurück.

Listing 7: find Befehl

```
1       private readonly Command _recordingList = Cli.Wrap("find") // updates list of
            recordings
2           .WithStandardOutputPipe(PipeTarget.ToDelegate(FileList))
3           .WithStandardErrorPipe(PipeTarget.ToDelegate(Console.WriteLine))
4           .WithArguments(RecordingDirectory);
```

Delete /api/recordings/ Dieser Endpoint erlaubt es einem User, beliebige Aufnahmedateien zu löschen.

Input: Die Methode akzeptiert eine Liste an Dateinamen.

Output: Der Endpoint gibt keine Daten zurück. Stattdessen verwendet er entweder den Content“Status-Code oder, falls beim Löschen der Dateien ein Fehler auftritt, einen „Bad Request“Status-Code.

Umsetzung: Das Programm führt einen einfachen rm (Remove) Befehl aus, um die angeführten Dateien nacheinander zu löschen.

Get /api/recordings/renameLast Dieser Endpoint wird benutzt, um Dateien umbenennen zu können. Die Idee war es, Aufnahmedateien benennen zu können, nachdem der Befehl, der sie erstellt hat, beendet wurde. Da diese Funktion eine mögliche Erweiterung aber keine Anforderung der Diplomarbeit war, wurde sie nicht vollständig implementiert, weshalb dieser Endpoint nicht genützt wird.

Input: Diese Methode benötigt genau einen String als Input. Dieser String ist der Name, den die Datei nach der Änderung haben soll.

Output: Diese Methode gibt entweder einen „No Content“Status-Code oder einen „Bad Request“Status-Code zurück.

Umsetzung: Das Programm überprüft als erstes, ob es eine Datei gibt, die umbenannt werden könnte. Falls keine Datei gefunden wird, schickt das Backend eine „Bad Request“als Antwort zurück. Falls das Programm eine passende Datei findet, wird diese mit dem mv Befehl umbenannt.

USBs

Get /api/usb/ Der RaspiSniffer verfügt über verschiedene Funktionen, die USB-Sticks benötigen. Das Betriebssystem des Raspberry Pi wurde so konfiguriert, dass USB-Geräte, die an den Raspberry Pi angeschlossen werden, automatisch gemountet werden. Damit das Frontend verbundene USB-Geräte anzeigen und der Nutzer diese verwenden kann, wird dieser Endpoint benötigt.

Input: Dieser Endpoint benötigt keine Input-Daten.

Output: Der Output ist eine Liste an Strings. Jeder dieser Einträge ist der Name eines USB-Geräts, das mit dem RaspiSniffer verbunden ist.

Umsetzung: Diese Funktion lässt sich auf drei eigenständige Teile aufteilen. Der erste Teil ist der Befehl, mit dem das Shell-Skript ausgeführt wird. Für diesen Schritt verwenden wir die CLI.Wrap-Bibliothek. Das Programm erstellt ein Command-Objekt, welches das richtige Skript ausführt. In diesem Fall speichert die Variable Tilde das Home-Directory des Linux-Benutzers, der auf dem Raspberry Pi angemeldet ist.

Listing 8: Command Objekt wird erstellt

```
1       Command _usbSticks = Cli.Wrap(Tilde + "/Scripte/USB/show_all_USB.sh")
2           // gets a list of USB sticks that are mounted on the Raspberry pi
3           .WithStandardOutputPipe(PipeTarget.ToDelegate(UsbShow)).
4           // PipeTarget.ToDelegate calls specified Function for each line that the command
                returns
5           WithStandardErrorPipe(PipeTarget.ToDelegate(Console.WriteLine));
```

Der Output des Shell-Skripts wird durch eine eigene Methode verarbeitet. Diese Methode erhält den Output des Command Zeile für Zeile. Da das Skript immer genau ein USB-Gerät pro Zeile ausgibt, muss diese Methode nur die Zeilen abspeichern.

Um die Daten formatiert an den Benutzer zurückzugeben, wandelt diese Methode die Liste in ein Array um. Aus diesem Array kann dann ein JSON-String erstellt werden, der danach dem Nutzer geschickt werden kann.

Listing 9: JSON-String wird erstellt und zurückgegeben

```
1       [HttpGet("/api/usb/")]
2       public async Task<IActionResult> GetUsbSticks()
3           // returns mounted USB sticks
4       {
5           Console.WriteLine("Getting USB-sticks");
6           _usbStickList = new List<string>();
7           await _usbSticks.ExecuteAsync();
8           return Ok(JsonSerializer.Serialize(_usbStickList.ToArray()));
9       }
```

Post /api/usb/ Dieser Endpoint trennt die Verbindung zwischen Raspberry Pi und USB-Stick. Der User kann einen oder auch mehrere USB-Sticks auf einmal auswerfen. Das Backend verwendet dafür den Linux-Befehl „umount. Dieser wird genützt, um Datenträger aus dem System auszuwerfen. Nach dieser Operation kann der USB-Stick problemlos entfernt werden.

Input: Eine Liste an USB-Geräten, die vom Raspberry Pi unmountet werden sollen. Um die genauen Namen der Geräte zu bekommen, sollte der Get /api/usb/ Endpoint aufgerufen werden.

Output: Dieser Endpoint gibt, wenn der unmount Befehl erfolgreich ist, einen leeren „No Contentt“zurück. Falls der Befehl nicht erfolgreich beendet wird oder ein Fehler in den Eingabewerten entdeckt wird, schickt das Backend einen „Bad Request“mit passender Nachricht zurück.

Umsetzung: Als erstes überprüft das Programm, ob die Eingabedaten richtig sind. Dafür sucht der Raspberry Pi nach allen verbundenen USB-Geräten und überprüft, ob das Gerät, das entfernt werden soll, in dieser Liste vorkommt. Falls es nicht gefunden wird, gibt die API eine „Bad Request“, mit passender Nachricht zurück. Falls kein Fehler gefunden wird, beginnt das Programm damit, die USB-Geräte auszuwerfen. Dafür wird für jedes angegebene Gerät ein „umountBefehl erstellt und ausgeführt. Nachdem die Geräte ausgeworfen wurden, überprüft das Programm, ob die Befehle erfolgreich waren. Es aktualisiert die Liste an verbundenen Geräten und kontrolliert, ob alle Geräte die entfernt werden sollten, aus dieser Liste verschwunden sind.

Listing 10: Backend überprüft Eingabe und führ umount aus

```
1       [HttpPost("/api/usb/")]
2       public async Task<IActionResult> DismountUsb([FromBody] string[] removableSticks)
3       // dismounts all selected USBs
4       {
5           _usbStickList = new List<string>();
6           await _usbSticks.ExecuteAsync();    // refresh USB list to confirm input
7           foreach (string stick in removableSticks)
8           {
9               if (!_usbStickList.Contains(stick))
10                  return BadRequest("one or all of USB sticks not found");
11          }
12
13          foreach (var removableStick in removableSticks)
14          {
15              await Cli.Wrap("umount") // unmount each USB
16                  .WithArguments(UsbPath + removableStick)
17                  .WithStandardErrorPipe(PipeTarget.ToDelegate(Console.WriteLine))
18                  .ExecuteAsync();
19          }
20
21          _usbStickList = new List<string>();
22          await _usbSticks.ExecuteAsync(); // refresh USBs again to confirm successful
                unmount operation
23
24          foreach (string stick in removableSticks)
25          {
26              if (_usbStickList.Contains(stick))
27                  return BadRequest("one or all of USB sticks not dismounted correctly");
28          }
29
30          return NoContent();
31      }
```

Post /api/usbcpy/ Dieser Endpoint erlaubt es einem User, Dateien vom RaspiSniffer auf einen angeschlossenen USB-Stick zu kopieren. Die Dateien, die zur auswahl stehen, sind die Aufnahme Dateien die durch den Befehl netsniff-ng entstehen.

Input: Dieser Endpoint benötigt ein String-Array, welches mindestens drei Einträge besitzt. Dieses Array besteht aus folgenden Werten:

- Dem Namen des USB-Sticks, der Ziel des Kopiervorgangs ist.
- Dem Pfad, unter dem die Daten auf dem USB-Gerät gespeichert werden sollen.
- Dem Namen von mindestens einer Aufnahmedatei die kopiert werden soll
- Das Array kann um beliebig viele Strings erweitert werden, wenn der User mehrere Dateien kopieren will

Output: Wenn der Kopiervorgang ohne Fehler durchgeführt wurde, gibt der Endpoint keine Daten zurück. Stattdessen wird nur ein „No Contentt“Status geschickt. Falls in den Inputs ein Fehler gefunden wird, oder falls der Kopier-Befehl fehlschlägt, gibt der Endpunkt stattdessen einen Fehlercode mit passender Nachricht zurück.

Umsetzung: Um die Eingabewerte zu überprüfen, wird als erstes kontrolliert, ob überhaupt ausreichend Werte vorhanden sind. Falls der Nutzer nur zwei oder weniger Inputs definiert, gibt das Programm sofort einen „Bad Request“Fehlercode zurück. Falls genügend Eingaben vorhanden sind, werden die angeschlossenen USB-Geräte neu geladen. Danach überprüft das Programm, ob das Ziel des Kopiervorgangs verbunden ist. Wenn der USB-Stick vorhanden ist, wird jede Datei, die kopiert werden soll, überprüft. Falls eine angegebene Datei nicht im Raspberry Pi vorhanden ist, wird die Funktion abgebrochen. Nur falls alle Inputs diesen Vorgaben entsprechen, werden Daten wirklich verändert. Der erste Schritt, den das Programm durchführt, ist ein "mkdir"Befehl. Dieser Befehl erstellt den Ordner, in dem der User die Aufnahmedateien speichern will. Nachdem der Ordner erstellt wurde, werden alle definierten Dateien nacheinander kopiert. Falls in diesem Vorgang ein Fehler entsteht, wird ein Fehlercode an den Nutzer zurückgeschickt. Wenn der Befehl ohne Fehler alle Dateien kopiert, gibt der Endpoint einen „No Contentt“Statuscode zurück.

Get /api/stopcpy/ Falls es bei dem Kopiervorgang zu einem Fehler kommt oder falls er zu lange dauert, kann dieser Endpunkt aufgerufen werden, um ihn zu unterbrechen. Dieser Abbruch beschädigt weder das Filesystem des Raspberry Pi noch das Filesystem des USB-Geräts.

Input: Diese Methode benötigt keine Inputdaten.

Output: Dieser Endpunkt sendet immer einen „No Contentt“Statuscode als Antwort. Nur wenn der Thread, in dem die Anfrage verarbeitet wird, abstürzt, kann eine andere Antwort entstehen.

Umsetzung: Um den Kopiervorgang abzubrechen, wird ein CancellationToken verwendet. Dieser ist in dem Singleton "Datastore"gespeichert und wird für alle Kopiervorgänge verwendet. Deshalb bricht diese Methode alle laufenden Kopiervorgänge auf einmal ab. Da ein Abgebrochener CancellationToken nicht mehr verwendet werden kann, um neue Befehle zu starten, erstellt der Datastore automatisch eine neue CancellationTokenSource, falls ein neuer benötigt wird.

Listing 11: CancellationToken wird aktiviert wodurch der Befehl abbricht

```
1       [HttpGet("/api/stopcpy/")]
2       public IActionResult StopUsbCopy()
3           // force-cancels copying to USB
4       {
5           Datastore.GetInstance().GetFileTransferCancelSource().Cancel();
6           return NoContent();
7       }
```

Speicherkapazität

Get /api/totalStorage Damit das Frontend die Speicherkapazität des Raspberry Pi anzeigen kann, wird dieser Endpoint zur Verfügung gestellt. Dieser Prozess ist in Linux basierten Betriebssystemen, dank dem df Befehl, sehr einfach. Dieser Befehl liefert Informationen über die Speicherkapazität, die dem System zur Verfügung steht.

Input: Diese Funktion benötigt keine Input Daten

Output: Diese Funktion liefert, wenn keine Probleme auftreten, einen String als Ergebnis. Dieser String ist die Anzahl an KiBs die dem RaspiSniffer zur Verfügung stehen.

Umsetzung: Die Ausgabe des df Befehl ist schwer auszulesen. Um dieses Problem zu lösen, verwenden wir die Befehle grep und cut. Mit grep wird die Zeile ausgewählt, die Daten über das richtige Filesystem enthalten. In dieser Partition werden alle Daten gespeichert, die nicht für das Betriebssystem wichtig sind. Mit dem Befehl cut werden alle Spalten, die nicht benötigt werden, aus der Ausgabe gefiltert. Diese formatierte Ausgabe ist einfach für das System zu lesen. Er muss nur von einem String zu einem Integer konvertiert werden.

Listing 12: Erstellung und Ausführung des df Befehls

```
1       [HttpGet("/api/totalStorage")]
2       public async Task<IActionResult> GetTotalStorage()
3       // calculates and returns total storage space of Raspberry Pi
4       {
5           if (_storageSpace == 0) // if storage has already been calculated we don't need to
                calculate it a second time
6           {
7               Command getStorage = Cli.Wrap("df").WithArguments(args => args
8                   // gets storage space of each Filesystem
9                   .Add("--output=source,size")
10              ) | Cli.Wrap("grep").WithArguments(args => args // filters correct filesystem
11                  .Add("-E")
12                  .Add("/dev/mmcblk0p2")
13              ) | Cli.Wrap("cut").WithArguments(args => args // deletes first column
14                      .Add("-d")
15                      .Add(" ")
16                      .Add("-f2-")
17                  ).WithStandardOutputPipe(PipeTarget.ToDelegate(MaxStorageSpaceFunc))
18                  .WithStandardErrorPipe(PipeTarget.ToDelegate(Console.Error.WriteLine));
19              try
20              {
21                  await getStorage.ExecuteAsync();
22              }
23              catch (CommandExecutionException e)
24              {
25                  Console.WriteLine(e.Message);
26                  return BadRequest();
27              }
28          }
29
30          return Ok(_storageSpace);
31      }
```

Weil sich der Speicherplatz, der dem RaspiSniffer zur Verfügung steht, sich nicht oft Verändert speichert das Backend die Größe des Speicherplatzes, nachdem diese berechnet wurde, und gibt auf weitere Anfragen diesen Wert zurück.

Get /api/usedStorage Dieser Endpoint gibt den derzeitigen Speicherverbrauch zurück. Er wird genützt um dem User eine Übersicht über den Speicherverbrauch zu verschaffen.

Input: Dieser Endpoint benötigt keine Eingaben.

Output: Er gibt den derzeitig verwendeten Speicher, in KiBs, zurück.

Umsetzung: Dieser Endpoint funktioniert ähnlich wie der totalStorage Endpoint. Er verwendet dieselbe Formatierung, um die Ausgabe des df Befehls lesbar zu machen, und verwendet nur eine andere Option für den df Befehl. Außerdem speichert das Backend den erhaltenen Wert nicht, da sich dieser öfters verändert.

Netzwerke

Get /api/ip/ Um die IP-Adresse des eth0 Port in dem lokal gehosteten Frontend anzeigen zu können, muss das Backend diesen Endpoint implementieren. Durch diesen Endpoint wird die richtige IP-Adresse ermittelt und an das Frontend übermittelt.

Input: Dieser Endpoint benötigt keinen Input.

Output: Der Nutzer erhält einen einzelnen String, der die IP-Adresse enthält.

Umsetzung: Da der Datastore die gewollte IP-Adresse bereits speichert, fragt das Backend diese ab und schickt sie an das Frontend.

Listing 13: Get IP Methode

```
1       [HttpGet("/api/ip/")]
2       public IActionResult GetIp()
3       {
4           // returns IP of eth0
5           return Ok(Datastore.GetInstance().GetIp());
6       }
```

Get /api/wifi/ Dieser Endpoint startet einen Befehl, der alle WLAN-Netzwerke, die sichtbar und in Reichweite des RaspiSniffers sind, sucht und in eine Liste schreibt. Diese Liste besteht aus WIFI-Objekten die jeweils die SSID eines Netzwerks und den „connected“ Wert speichern. Dieser Wert zeigt, ob der Raspberry Pi mit einem bestimmten Netzwerk verbunden ist.

Obwohl mehrere WLAN Access Points dieselbe SSID haben können, wird jede SSID nur einmal gespeichert. Falls es für eine SSID mehrere Access Points geben sollte, werden alle zusammengefasst und falls der RaspiSniffer mit einem dieser Geräte verbunden ist, wird in dem zugehörigen Wifi Objekt „connected“ auf wahr gesetzt.

Input: Dieser Endpoint benötigt keine Eingabe.

Output: Der Output besteht aus einem Array an WIFI Objekten. Dieses Array wird in einen JSON-String konvertiert und dieser wird als Antwort an den Benutzer geschickt.

Umsetzung: Der Endpoint besteht aus drei Teilen. Dem nmcli Befehl, dem Delegate WifiListFunc und der wirklichen Funktion, die das Backend aufruft, wenn es eine Anfrage erhält.

Nmcli bietet verschiedene Argumente, die die Ausgabe vereinfachen. Das Backend verwendet -t, –colors und –escape. Diese formatieren die Ausgabe des Befehls, damit sie für ein Programm einfach zu lesen ist. Die Optionen –colors und –escape verändern die Ausgabe, sodass keine Escape-Zeichen, das sind Zeichen die verwendet werden um die Ausgabe zu formatieren, oder Farben verwendet werden.

Listing 14: nmcli konfiguration

```
1      private readonly Command _getWifiList = Cli.Wrap("nmcli").WithArguments( // updates
           basic list of available WIFIs
2          args => args
3              .Add("-t")
4              .Add("--colors")
5              .Add("no")
6              .Add("--escape")
7              .Add("no")
8              .Add("--fields")
9              .Add("SSID,IN-USE")
10             .Add("device")
11             .Add("wifi")
12             .Add("list")
13     )
14     .WithStandardOutputPipe(PipeTarget.ToDelegate(WifiListFunc))
15     .WithStandardErrorPipe(PipeTarget.ToDelegate(Console.WriteLine));
```

Dank der Argumente, des nmcli-Befehls, ist dessen Ausgabe sehr einfach zu verarbeiten. Die Spalten SSID und IN-USE sind immer durch einen Doppelpunkt getrennt. Deshalb kann die Funktion String.split() eine Zeile der Ausgabe in ihre einzelnen Felder zerlegen. Diese werden

dann in ein neues WIFI-Objekt gespeichert. Um richtig abzuspeichern, ob der Raspberry Pi mit einem bestimmten WLAN verbunden ist, überprüft das Programm, ob die Zeile nach dem Doppelpunkt leer ist. Nur falls eine Verbindung besteht, ist die zweite Spalte nicht leer.

Listing 15: WIFI Delegate Funktion

```
1       private static readonly Func<string, CancellationToken, Task> WifiListFunc = (line,
            token) =>
2       {
3           try
4           {
5               if (line.Split(':').Length < 2 ||
                    line.Split(':')[0].Trim().Equals(string.Empty))
6                   // if first string is empty the raspberry pi is picking up a hidden Wi-Fi
7                   return Task.CompletedTask;
8               string newSsid = line.Split(':')[0]; // first string is the SSID
9               bool newIsConnected = !line.Split(':')[1].Trim().Equals(String.Empty);
10              // if second string is not empty the Raspberry Pi is connected to that Wi-Fi
11              foreach (Wifi wifi in _wifis) // checks if Wi-Fi already has SSID in List
12              {
13                  if (wifi.Ssid.Equals(newSsid))
14                  {
15                      if (wifi.IsConnected)
16                      {
17                          return Task.CompletedTask;
18                      }
19                      wifi.IsConnected = newIsConnected;
20                      // if the new SSID is already in the list but the old one is not
                            connected while this one is connected mark old SSID as connected
21                      // this is done to avoid confusion because multiple routers can share
                            the same SSID and we can only be connected to one router
22                      return Task.CompletedTask;
23                  }
24              }
25              _wifis.Add(new Wifi(newSsid, newIsConnected));
26          }
27          catch (Exception e)
28          {
29              Console.WriteLine("Error while scanning for available Wifi connections " + e);
30          }
31          return Task.CompletedTask;
32      };
```

Die Funktion, die auf Anfrage an den Endpoint aufgerufen wird, startet den nmcli-Befehl und gibt das Ergebnis zurück.

Post /api/wifi/ Dieser Endpoint wird verwendet, um genauere Informationen über ein bestimmtes WLAN-Netzwerk zu bekommen. Die wichtigste dieser Informationen ist der sogenannte Security Type. Diese wird vom Frontend verwendet, damit ein Benutzter, alle benötigten Daten eingeben kann. Obwohl meistens für so einen Fall eine Get Methode verwendet wird, ist dieser Endpoint eine Post Methode. In diesem Fall wird ein Body-Element benötigt, was in einer Get Methode in dem ASP.NET Core Framework nicht zur Verfügung steht. Das Body-Element wird unbedingt benötigt, da eine SSID bestimmte Sonderzeichen enthalten kann, die nicht in einer URL verschickt werden können.

Input: Dieser Endpoint benötigt die SSID des WLANs, das untersucht werden soll.

Output: Die Antwort besteht aus einem einzelnen WIFI Objekt, welches zu einem JSON-String konvertiert wurde.

Umsetzung: Die Methode verwendet eine leicht veränderte Version des nmcli-Befehls, der für die get all Wifis Methode verwendet wird. Der große Unterschied zwischen den zwei Endpoints, ist die Spalte SSecurity", die hinzugefügt wurde. Die Ausgabe dieses Befehls wird zusätzlich durch die Befehle grep und head so formatiert, dass nur das gesuchte WLAN-Netzwerk gefunden wird. Aus den Daten wird ein neues WIFI-Objekt gebildet, welches als Antwort zurückgeschickt wird.

Put /api/wifi/connect Dieser Endpoint existiert, damit sich der Raspberry Pi mit einem beliebigen WLAN-Netzwerk verbinden kann. Er bietet Optionen für alle Arten von Sicherheitstypen. Außerdem kann er sich mit dem richtigen Input mit versteckten WLAN-Netzen verbinden.

Input: Dieser Endpoint benötigt ein String Array als Input, welches aus folgenden Teilen besteht.

- ein boolescher Wert der darstellt ob das Netzwerk, mit dem man sich verbinden will, versteckt ist oder nicht
- die SSID des Netzwerks
- (optional) das Passwort das verwendet werden soll
- (optional) der Benutzername der verwendet werden soll

Output: Dieser Endpoint gibt keine Daten als Antwort zurück. Stattdessen kommuniziert er mit dem Frontend durch Status-Codes. Falls die Operation gelingt, gibt der Endpoint einen „No Contentt“Status zurück. Falls ein Fehler passiert, oder der Benutzer zu viel oder zu wenig Argumente deklariert, gibt er stattdessen einen „Bad Request“Status-Code zurück.

Umsetzung: Der Verbindungsaufbau läuft über zwei Shell-Skripts. Eines dieser Skripte wird dazu genutzt, sich mit einem versteckten WLAN-Netzwerk zu verbinden. Das andere wird verwendet, um sich mit einem öffentlichen Netzwerk zu verbinden. Das Programm wählt, welches Skript es verwendet, durch den ersten Input-Wert.

Listing 16: richtiges Skript wird gewählt

```
1          Command connector;
2          if (UserData[0].Trim().ToLower().Equals("true"))
3          {
4              connector = Cli.Wrap(Tilde + "/Scripte/WIFI/Connect_to_hidden_Wifi.sh"); //
                   script that is used to connect to the Wi-Fi
5          }
6          else
7          {
8              connector = Cli.Wrap(Tilde + "/Scripte/WIFI/Connect_to_Wifi.sh"); // script
                   that is used to connect to the Wi-Fi
9          }
```

Danach werden die angegebenen Daten an das Skript weitergegeben. Falls nur eine SSID vorgegeben ist, muss das Programm einen Platzhalter als Passwort übergeben. Wenn alle Argumente definiert sind, führt das Backend den Befehl aus und gibt, falls kein Problem auftritt, den Status-Code „No Contentt“zurück.

Listing 17: Argumente werden definiert

```
1              if (userData.Length < 3) // if only SSID is passed to backend it's an open
                   Wi-Fi, so we use a placeholder as the password
2              {
3                  await connector.WithArguments(args => args
4                      .Add(userData[1])
5                      .Add("Passme123!")
6                  ).ExecuteAsync();
7                  return NoContent();
8              }
9
10             if (userData.Length < 4)  // if SSID and Password are passed to the backend we
                   use the script as intended
11             {
12                 await connector.WithArguments(args => args
13                     .Add(userData[1])
14                     .Add(userData[2])
15                 ).ExecuteAsync();
16                 return NoContent();
17             }
18
19             if (userData.Length < 5)
20                 // if SSID, Password and a username are passed to the backend, we send all
                       three to the script which will handle the different security
21             {
22                 await connector.WithArguments(args => args
23                     .Add(userData[1])
24                     .Add(userData[2])
25                     .Add(userData[3])
26                 ).ExecuteAsync();
27                 return NoContent();
28             }
```

Get /api/networks/ Das Frontend erlaubt es einem User auszuwählen, welches Netzwerk der RaspiSniffer abhören soll. Um diese Auswahl zu ermöglichen, muss das Frontend wissen, mit welchen Netzen der Raspberry Pi verbunden ist.

Input: Dieser Endpoint benötigt keine Inputs.

Output: Die Antwort besteht aus einer Liste an Strings. Jeder Eintrag steht für ein Netz, mit dem der RaspiSniffer verbunden ist. Diese Liste ist maximal zwei Einträge lang, da der Raspberry Pi sich mit nur einem WLAN-Netzwerk auf einmal verbinden kann und alle angeschlossenen Ethernet-Adapter als ein Netzwerk gewertet werden.

Umsetzung: Als erstes überprüft das Backend, ob sich der RaspiSniffer mit einem WLAN-Netzwerk verbunden hat. Falls dies der Fall ist, speichert es die SSID des Netzwerks. Um die angeschlossenen Ethernet-Adapter zu erkennen, lest der Raspberry Pi den Inhalt des Ordners /sys/class/net/. Dieses Verzeichnis speichert Informationen über alle Network Interfaces, die dem Betriebssystem zur Verfügung stehen. Aus diesem Verzeichnis kann man ablesen, wie viele Ethernet Anschlüsse auf dem RaspiSniffer vorhanden sind. Falls es mehr als nur den eth0 Anschluss gibt, ist der Raspberry Pi mit einem Ethernet Netzwerk verbunden und der Endpoint schickt diese Information an das Frontend mit.

Get /api/wifi/disconnect Dieser Endpoint wird genützt, damit sich der Raspberry Pi von einem WLAN-Netzwerk trennen kann. Er muss aufgerufen werden, bevor ein Nutzer versucht sich mit einem anderen WLAN zu verbinden.

Input: Dieser Endpoint benötigt keinen Input.

Output: Als Antwort schickt dieser Endpoint entweder einen „No Contentt“Status-Code oder einen „Bad Request“Status-Code.

Umsetzung: Das Backend führt ein bestimmtes Skript aus, welches den Raspberry Pi von allen WLAN-Verbindungen trennt. Falls ein Fehler auftritt, gibt das Programm einen „Bad Request“Fehler-Code zurück. Falls das Skript erfolgreich endet wird stattdessen ein „No Contentt“Status-Code verschickt.

Listing 18: Skript wird vorbereitet und ausgeführt

```
1       [HttpGet("/api/wifi/disconnect")]
2       public async Task<IActionResult> DisconnectFromWifi()
3       // Disconnects from all WIFIs
4       {
5           Command disconnect = Cli.Wrap(Tilde + "/Scripte/WIFI/Disconnect_from_Wifi.sh") //
                calls script to disconnect from Wi-Fi
6               .WithStandardErrorPipe(PipeTarget.ToDelegate(Console.Error.WriteLine))
7               .WithStandardOutputPipe(PipeTarget.ToDelegate(Console.WriteLine));
8           try
9           {
10              await disconnect.ExecuteAsync();
11          }
12          catch (CommandExecutionException e)
13          {
14              Console.WriteLine(e.Message);
15              Console.WriteLine(e.StackTrace);
16              return BadRequest();
17          }
18          return NoContent();
19      }
```

8.2.7 Publishing

Das .Net Framework wurde für das Betriebssystem Windows entwickelt. Um es trotzdem in einem Linux basierten System verwenden zu können, muss es mit dem dotnet publish CMD-Befehl für Linux kompiliert werden. Da der Raspberry Pi, auf dem diese Diplomarbeit gehostet wird, außerdem einen ARM-Prozessor hat, muss der Befehl zusätzlich angepasst werden. Mit der Option -r linux-arm64 wird das ausgewählte Projekt für einen Raspberry Pi vorbereitet. Die zweite Option, –self-contained false, wird benötigt weil diese Option in Linux Systemen oft Fehler verursacht.

```
dotnet publish -r linux-arm64 --self-contained false
```

Der Befehl erstellt einen Ordner, der die kompilierten Dateien enthält. Der Inhalt dieses Ordner kann auf verschiedene Arten gehostet werden. In dieser Diplomarbeit haben wird den server mit dem dotnet Befehl gestartet.

8.3 Frontend

8.3.1 Erstellung der Webapplikation

Das RaspiSniffer-Webinterface ist eine Single Page Application, diese wurde mittels einer Blazor WebAssembly verwirklicht. Der Grund dafür ist, dass Blazor WebAssembly die Webseite im Browser des Clients rendert und nicht am Server. Dadurch wird dem Raspberry Pi ein wenig Arbeit abgenommen. [28] [28]

Die Benutzeroberfläche ist so aufgebaut, dass sie auf einem herkömmlichen 16:9-Bildschirm für Desktopanwendungen und einem 4:3-Bildschirm für Raspberry Pis leichte visuelle und funktionale Unterschiede gibt.

Die Benutzeroberfläche besteht aus drei Seiten. Teile der Logik sind außerdem in separate Komponenten ausgelagert.

Die Erstellung des Projekts wurde mit JetBrains Rider umgesetzt. Der erste Schritt war, Rider zu öffnen und eine neue Solution zu erstellen. Im Nächsten wurden der Projektname und der Solutionname eingegeben. Als .NET-Version wurde „.NET 8“ verwendet und als Programmiersprache C#. Danach wurde „Blazor WebAssembly Standalone App“ als Template ausgewählt.

Nach alledem war das Projekt erstellt und geöffnet. In dem Projekt befanden sich aber Klassen, Ordner und Bilder, die nicht benötigt wurden. Diese wurden gelöscht, und übrig blieb die Klasse MainLayout.razor, die so angepasst wurde, dass sie keinen Fehler mehr verursachte.

Listing 19: MainLayout.razor Klasse

```
1 @inherits LayoutComponentBase
2
3 @Body
```

Zu guter Letzt wurde eine Ordnerstruktur aufgebaut. Zu den nun bestehenden Ordnern wurden die Ordner, „Components“, „Model“ und „Service“ erstellt. In dem Components-Ordner wurden dann nochmal eigene Ordner für „General“, „Homescreen“, „Recordings“ und „WIFI“ unterteilt. Weiter wurde noch ein images-Ordner in wwwroot erstellt.

[28] *https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-9.0*

8.3.2 Generelle Komponenten

Die generellen Komponenten sind Komponenten, die von mehr als nur einer Seite verwendet werden. Diese sind im Ordner Components/General gespeichert.

Informations-Pop-Up

Das Informations-Pop-Up, Klasse InformationPopUp.razor, wird angezeigt, wenn dem Nutzer eine Information mitgeteilt werden soll. Dabei wird der Hintergrund grau und es wird ein kleines Pop-Up im Vordergrund angezeigt. Dieses enthält eine Nachricht und einen Knopf, mit dem der Nutzer das Pop-Up schließen kann.

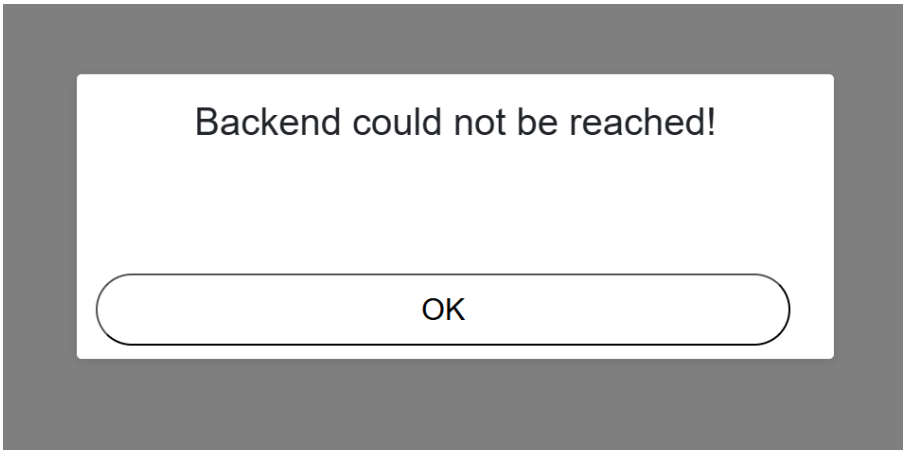

Abbildung 26: Informations-Pop-Up

Umsetzung: Da das Pop-Up als Komponente in den Seiten eingebunden ist, soll dieses nur dann angezeigt werden, wenn es auch sinnvoll ist. Um die Funktionalitäten des Pop-Ups zu verwirklichen, wird im HTML der Komponente (siehe Listing 20) das oben Beschriebene erstellt.

Listing 20: Informations-Pop-Up HTML

```
@if (isVisible)
{
    <div class="popup-overlay">
        <div class="popup">
            <div class="message">@Message</div>
            <button class="button" @onclick="CloseAndInvoke">OK</button>
        </div>
    </div>
}
```

Die Sichtbarkeit hängt nun von einer Abfrage eines Booleans im Code ab, in dieser Form wird die Sichtbarkeit eines Pop-Ups in jeder anderen Pop-Up-Komponente implementiert. Diese Variable ist standardmäßig „false“. Der Boolean wird bei den Methoden auf „true“ gesetzt (siehe Listing 21), um das Pop-Up zu öffnen und wieder auf „false“ gesetzt (siehe Listing 22), um das Pop-Up wieder zu schließen.

Listing 21: Informations-Pop-Up Show() Methode

```
public void Show()
{
    isVisible = true;
    StateHasChanged();
}
```

Listing 22: Informations-Pop-Up Close() Methode

```
public void Close()
{
    isVisible = false;
    StateHasChanged();
}
```

Dabei sorgt StateHasChanged() dafür, dass die Komponente über Änderungen informiert wird. Diese Methode wird in den anderen Komponenten auch verwendet.

Damit eine Nachricht dynamisch mitgegeben werden kann, wird die Variable so erstellt:

Listing 23: Parameter Implementierung Variable

```
[Parameter] public string Message { get; set; }
```

Somit erfolgt die Erstellung der Komponente wie folgt:

Listing 24: Parameter Implementierung in Erstellung der Komponente

```
<InformationPopUp @ref="informationPopUp" Message="@message"></InformationPopUp>
```

[Parameter] gibt einen Parameter an, der bei der Erstellung der Komponente mitgegeben werden kann. Diese Lösung wird bei weiteren Implementierungen von Parameter-Übergaben bei Komponenten eingesetzt.

Das Schließen des Pop-Ups erfolgt über den Ok-Button, dieser ist mit der Methode CloseAndInvoke() (siehe Listing 25) verbunden.

Listing 25: Ok-Button onclick Methode

```
private async Task CloseAndInvoke()
{
    await OnOk.InvokeAsync();
    Close();
}
```

Dabei steht die Methode im button-Element im „@onclick“.

Cancellation-Pop-Up

Das Cancellation-Pop-Up, Klasse CancellationPopUp.razor, wird angezeigt, wenn das Programm einen Prozess durchführt, auf den gewartet werden muss, jedoch auch abgebrochen werden kann. Dabei wird der Hintergrund grau und es wird ein kleines Pop-Up im Vordergrund angezeigt. Dieses enthält eine Nachricht und einen Knopf, mit dem der Nutzer den Prozess abbrechen und das Pop-Up schließen kann.

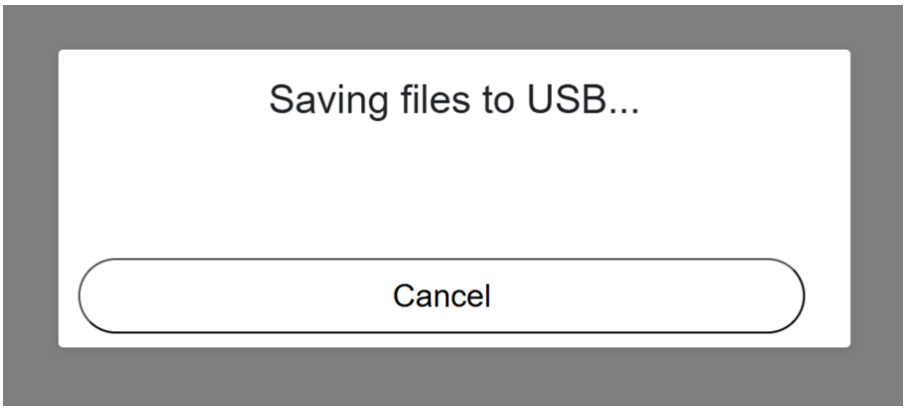

Abbildung 27: Cancellation-Pop-Up

Umsetzung: Da das Pop-Up als Komponente in den Seiten eingebunden ist, soll dieses nur dann angezeigt werden, wenn es auch sinnvoll ist. Um die Funktionalitäten des Pop-Ups zu verwirklichen, wird im HTML der Komponente (siehe Listing 26) das oben Beschriebene erstellt.

Listing 26: Cancellation-Pop-Up HTML

```
@if (isVisible)
{
    <div class="popup-overlay">
        <div class="popup">
            <div class="message">@Message</div>
            <button class="button" @onclick="CloseAndInvoke">Cancel</button>
        </div>
    </div>
}
```

Implementierung der Sichtbarkeit und der Nachricht, wie bei Informations-Pop-Up.

Damit beim Drücken des Cancel-Buttons mehr passiert, als das Schließen des Pop-Ups, wird die Variable OnClose (siehe Listing 27) definiert.

Listing 27: EventCallback Variable

```
[Parameter] public EventCallback OnClose { get; set; }
```

Dieser Variable kann bei der Erstellung eine Methode übergeben werden. Somit erfolgt die Erstellung der Komponente wie folgt:

Listing 28: EventCallback Implementierung in Erstellung der Komponente

```
1 <CancellationPopUp @ref="cancellationPopUp" Message="Saving files to USB..."
      OnClose="HandleCancelUsb"></CancellationPopUp>
```

In dem Fall wird die Methode „HandleCancelUsb“ ausgeführt, falls der Cancel-Button gedrückt wird. Diese Lösung wird bei weiteren ähnlichen Implementierungen verwendet.

8.3.3 Homescreen

Page

Der Homescreen, Klasse Home.razor, ist die Hauptseite der RaspiSniffer-Weboberfläche. Die Seite beinhaltet eine Box, den sogenannten Sniff-Log, die die bearbeiteten Pakete aus dem Backend anzeigt. Der Homescreen beinhaltet fünf Buttons:

- Start: Der Start-Button öffnet das Recordings-Pop-Up, in dem Anpassungen für die Aufzeichnung gemacht werden können. Während einer Aufzeichnung ändert sich der Text auf „Stop“.
- Filter: Der Filter-Button klappt eine Menü auf, in dem der Benutzer, Filter für die Aufzeichnung auswählen kann. Dabei kann es sich um bestimmte Pakete oder Ports handeln.
- Clear: Der Clear-Button leert den Sniff-Log und setzt den Zeitpunkt des Sniff-Logs auf die aktuelle Zeit, damit die vom Backend abgefragten Pakete die vor dem gespeicherten Zeitpunkt erstellt wurden, nicht mehr in den Sniff-Log geladen werden.
- WIFI: Der WIFI-Button führt den Benutzer auf die WIFI-Seite, auf der die Netzwerkeinstellungen des RaspiSniffers bearbeitet werden können.
- Recordings: Der Recordings-Button führt den Benutzer auf die Recordings-Seite, auf der der Benutzer Files auf das eigene Gerät herunterladen, auf einen USB-Stick kopieren oder löschen kann.

Weiter gibt es zwei Labels, die Informationen zum RaspiSniffer anzeigen:

- Adresse: Die Adresse zeigt an über welche Webadresse die Webseite aufgerufen werden kann. Dieses Label wird nur beim 4:3-Layout angezeigt. Grund dafür ist, dass die IP-Adresse nicht an einem PC angezeigt werden muss, da die Adresse in der Adress- und Suchleiste steht, aber auf dem Raspberry Pi Bildschirm schon, damit der Benutzer die Weboberfläche am PC finden kann.

- Speicherplatz: Der Speicherplatz des Raspberry Pi steht auf der Weboberfläche geschrieben, damit der Benutzer weiß wie viel Speicherplatz noch nicht belegt ist und so abschätzen kann, ob genug Platz für eine weitere Aufzeichnung frei ist.

Die Hauptseite hat zwei verschiedene Layouts, eine für einen 4:3-Bildschirm, Raspberry Pi-Bildschirm, und eine für einen 16:9-Bildschirm, PC-Bildschirm:

16:9-Layout: Beim 16:9-Layout sind die Buttons am unteren Rand platziert und der Sniff-Log belegt die oberen 75 Prozent der Seite. Der Speicherplatz des RaspiSniffer steht am rechten unteren Rand des Sniff-Logs geschrieben. Die Webadresse wird nicht angezeigt. Pakete können im Sniff-Log vollständig angezeigt werden.

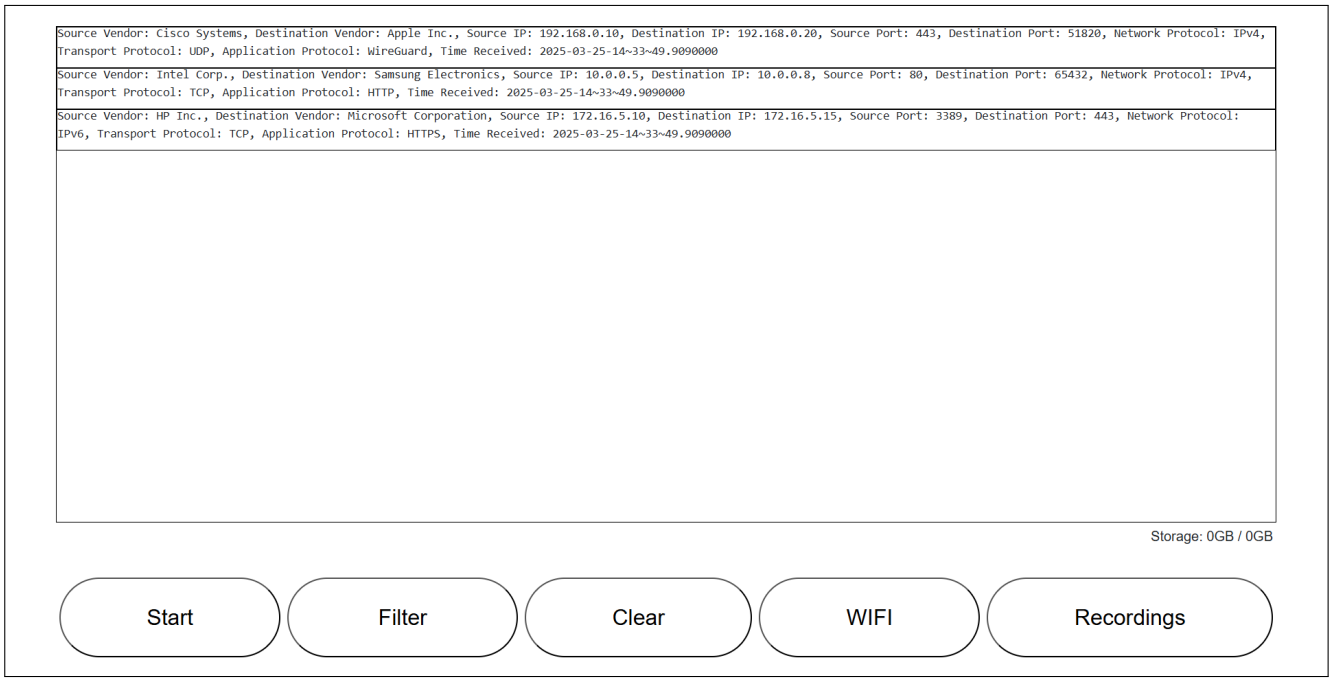

Abbildung 28: 16:9-Bildschirm Homescreen

4:3-Layout: Beim 4:3-Layout sind die Buttons am rechten Rand platziert. Der Sniff-Log belegt die linken 55 Prozent der Seite. Die Webadresse steht in der oberen rechten Ecke der Weboberfläche geschrieben. Der Speicherplatz steht unter der Webadresse. Die Pakete können im Sniff-Log nicht voll angezeigt werden, ohne die Lesbarkeit zu behindern. Diese Entscheidung wurde getroffen in der Annahme, dass der Bildschirm in etwa die Größe des Raspberry Pis hat.

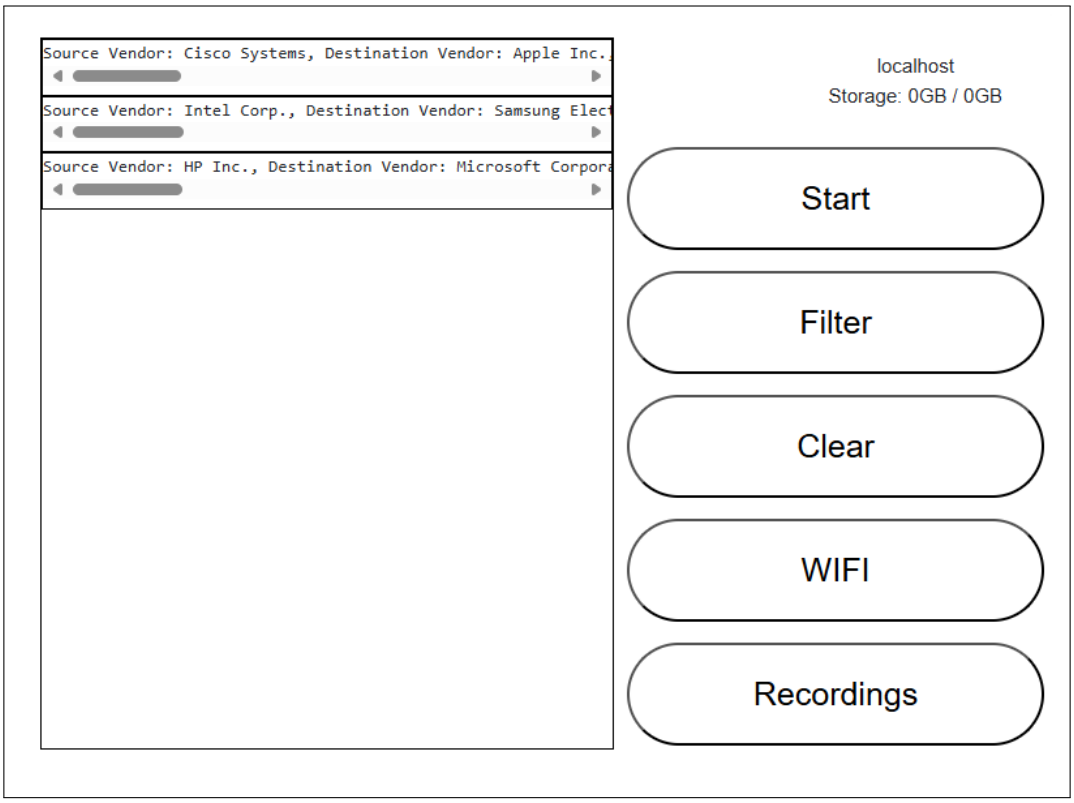

Abbildung 29: 4:3-Bildschirm Homescreen

Umsetzung: Da diese Klasse eine Seite ist, soll sie mittels einer URL erreichbar sein. Dies wird gemacht, indem die folgende Zeile eingebaut wird:

Listing 29: @page Implementierung

```
1  @page "/"
```

Die Implementierung der Seitenerreichbarkeit erfolgt bei den anderen Seiten auf dieselbe Weise. Da für das Nutzen der Seite verschiedene Komponenten und Services benötigt werden, müssen diese importiert und injiziert werden. Dies wird gemacht, indem die folgenden Zeilen implementiert werden:

Listing 30: Imports und Injects

```
@using RaspiSnifferWebpage.Components.Homescreen
@using RaspiSnifferWebpage.Components.General
@using RaspiSnifferWebpage.Service
@inject NavigationManager NavigationManager
@inject ISniffLogService SniffLogService
@inject IApiService ApiService
```

Die Implementierung der Imports und Injects erfolgt bei den anderen Seiten und Komponenten genauso. Zuallererst soll der Titel der Seite geändert werden, das wurde mit diesem HTML-Tag umgesetzt:

Listing 31: Page Title Homescreen

```
<PageTitle>Homescreen</PageTitle>
```

Dabei wird der Titel der Seite auf „Homescreen“ gesetzt. Diese Implementierung wird bei den anderen Seiten genauso eingesetzt.

Im nächsten Schritt werden die Komponenten in die Seite eingefügt. Diese werden im HTML folgendermaßen implementiert:

Listing 32: Komponenten im Homescreen

```
<div class="sniff-log-container">
    <SniffLog @ref="sniffLog"></SniffLog>
</div>
<RecordingsConfigurationPopUp @ref="recordingsPopup"></RecordingsConfigurationPopUp>
<InformationPopUp @ref="informationPopUp" Message="@message"></InformationPopUp>
```

Weitere Implementierungen von Komponenten funktionieren genauso. Die Komponente SniffLog ist in einem eingebaut, da ein einfach formatiert werden kann.

Da das Filter-Drop-Down kein eigenes Pop-Up ist, wird es eigens im HTML angelegt, dies wurde mit folgendem HTML implementiert:

Listing 33: Implementation Filter Button HTML

```
<div class="dropup">
    <button class="button" @onclick="ToggleFilterDropdown">Filter</button>
    @*A drop-up menu in which you can define filters for the sniffing*@
    <div class="dropup-menu" style="display: @(filterDropdownOpen ? "block" : "none")">
        @foreach (var filter in filters)
        {
            <div class="dropup-item">
                <label>
                    <input type="checkbox" @bind="filter.IsChecked"/>
                    @filter.Name
                </label>
            </div>
        }
        <div class="dropup-item">
            <label for="port" class="port-label">Port:</label>
            <input type="number" id="port" class="port-field" @bind="portFilter"/>
        </div>
    </div>
</div>
```

Ob das Filter-Drop-Down angezeigt wird, wird über den Boolean „filterDropdownOpen“ definiert. Bei „true“wird es angezeigt, bei „false“ nicht. Diese Variable ist standardmäßig „false“.

Beim Drücken des Buttons wird die Methode ToggleFilterDropdown() ausgeführt. Diese ändert den Wert von filterDropdownOpen zum gegenteiligen Wert.
Wenn die Seite nun initialisiert wird, werden im Hintergrund Timer gestartet. Dazu wird auch noch überprüft, ob der Zeitpunkt gesetzt werden muss. Wenn das der Fall ist, wird das gemacht. Die Timer rufen jeweils in einem festen Zeitintervall eine selbst kreierte Methode und StateHasChanged() asynchron auf (siehe Listing 34).

Listing 34: OnInitialized Homescreen

```
protected override void OnInitialized()
{
    textTimer= new Timer(async _ =>
    {
        ToggleStartStopText();
        StateHasChanged();
    }, null, 0, 100);

    timer = new Timer(async _ =>
    {
        await LoadStorage();
        StateHasChanged();
    }, null, 0, 15000);

    if (SniffLogService.LastClearedTimestamp.Year == 1)
    {
        SniffLogService.LastClearedTimestamp = DateTime.Now;
    }
}
```

Bei der Methode ToggleStartStopText() wird auf Grund des Booleans „isRunning“ im Sniff-Log-Service entschieden, ob der Text „Stop“ oder „Start“in den String „startStopText“ eingefügt wird.

Listing 35: Raspberry Pi benutzter und besetzter Speicher Methode

```
private async Task LoadStorage()
{
    usedStorage = await ApiService.GetUsedStorage();
    usedStorage = usedStorage * 1024 / 1000000000;

    totalStorage = await ApiService.GetTotalStorage();
    totalStorage = totalStorage * 1024 / 1000000000;
}
```

Bei der Methode LoadStorage() (siehe Listing 35) wird das API-Service benutzt, um mit der API zu kommunizieren. Die Anfragen werden, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Die in KiB erhaltenen Zahlen werden in GB umgerechnet.

Weiter gibt es noch Methoden für die restlichen Buttons.

Listing 36: Start-/Stop-Button Methode

```
private async void ToggleStartStop()
{
    if (SniffLogService.isRunning)
    {
        sniffLog.StopRecording();
    }
    else
    {
        filterDropdownOpen = false;
        await recordingsPopup.ShowPopup(GetFilterString());
    }
}
```

In der Methode ToggleStartStop() (siehe Listing 36) wird abgefragt, ob gerade eine Aufzeichnung läuft. Falls das der Fall ist, wird die Aufzeichnung gestoppt. Falls das nicht der Fall ist, wird das Filter-Drop-Down geschlossen und das Recordings-Pop-Up geöffnet, dabei wird auch ein String bestehend aus den ausgewählten Filtern mitgeschickt.

Listing 37: Clear-Button Methode

```
private void ClearPackets()
{
    sniffLog.ClearPackets();
    SniffLogService.LastClearedTimestamp = DateTime.Now;
}
```

Die Methode ClearPackets() (siehe Listing 37) ruft die ClearPackets() Methode im Sniff-Log auf. Um nun keine alten Pakete mehr anzuzeigen, wird der Zeitpunkt im Sniff-Log-Service auf den gegenwärtigen Moment gesetzt.

Listing 38: WIFI-Button Methode

```
private void NavigateToWiFi()
{
    timer.Dispose();
    textTimer.Dispose();
    sniffLog.GetTimer().Dispose();

    NavigationManager.NavigateTo("/wifi");
}
```

Bei der Methode NavigateToWiFi() (siehe Listing 38) werden die Timer beendet und es wird zu „/wifi“ weitergeleitet. Diese Implementierung findet auch bei den anderen Weiterleitungen statt. Somit auch in diesem Fall beim Recordings-Button. Mit dem einzigen Unterschied, dass dieser zu „/recordings“ weiterleitet.

Sniff-Log

Der Sniff-Log, Klasse SniffLog.razor, ist die Komponente, die die Pakete in aufbereiteter Form anzeigt. Dieser zeigt nur Pakete an, die nach dem für die Session gespeicherten Zeitpunkt erstellt wurden. Dabei werden die aktuellsten Pakete unten angezeigt.

Damit der Nutzer nicht andauernd nach unten scrollen muss, um die aktuellsten Pakete zu sehen, wird das automatisch gemacht, solange eine Aufzeichnung läuft.

Falls gerade eine Aufzeichnung geplant ist, werden Texte im Sniff-Log angezeigt. Als Erstes ein Text, der den Nutzer darüber informiert, danach ein Text, der den Startzeitpunkt anzeigt und zum Schluss ein Text, der den Endzeitpunkt anzeigt.

Der Sniff-Log hat zwei verschiedene Layouts, eine für einen 4:3-Bildschirm, Raspberry Pi-Bildschirm, und eine für einen 16:9-Bildschirm, PC-Bildschirm:

16:9-Layout: Beim 16:9-Layout ist der Sniff-Log ein horizontales Rechteck, wobei die Informationen der Pakete mit einem Zeilenumbruch vollständig angezeigt werden.

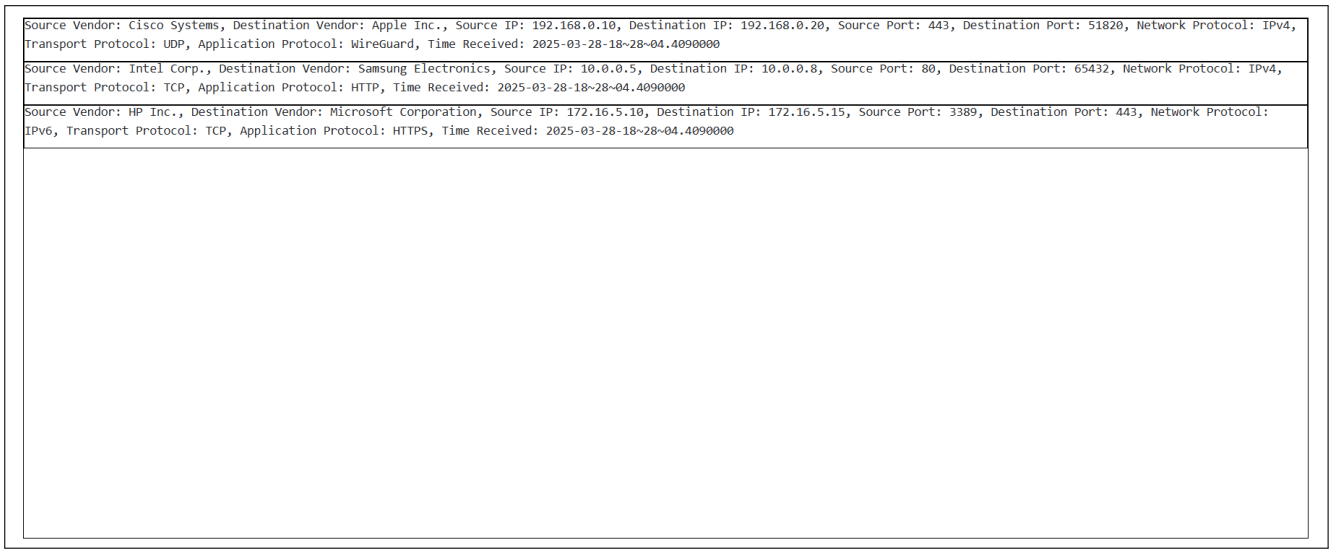

Abbildung 30: 16:9-Bildschirm Sniff-Log

4:3-Layout: Beim 4:3-Layout ist der Sniff-Log ein vertikales Rechteck, wobei die Informationen der Pakete ohne einen Zeilenumbruch in einem horizontalen Scroll-Element angezeigt werden.

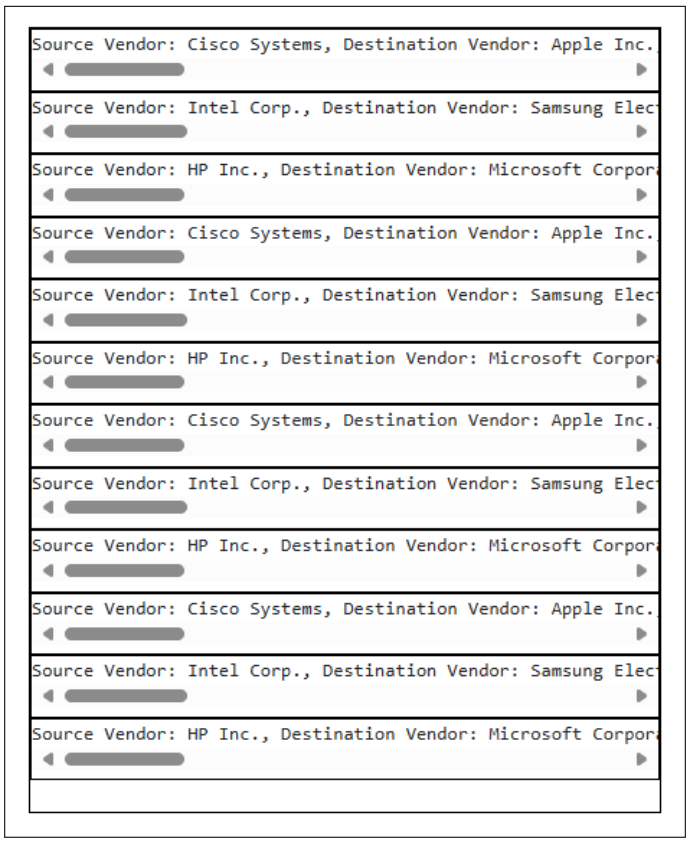

Abbildung 31: 4:3-Bildschirm Sniff-Log

Die Informationen der Pakete, die angezeigt werden, sind:

- Source Vendor: Der Hersteller der Quelle.
- Destination Vendor: Der Hersteller des Ziels.
- Source IP: Die IP-Adresse des sendenden Geräts.
- Destination IP: Die IP-Adresse des empfangenden Geräts.
- Source Port: Der Quellport.
- Destination Port: Der Zielport.
- Network Protocol: Das verwendete Netzwerkprotokoll.
- Transport Protocol: Das verwendete Transportprotokoll
- Application Protocol: Das verwendete Applikationsprotokoll
- Time Received: Der Zeitpunkt, an dem das Paket empfangen wurde.

Umsetzung: Zuerst werden Imports und Injects gemacht, diese erfolgen wie im Homescreen. Um die Pakete und die geplante Aufzeichnung anzuzeigen, wird im HTML (siehe Listing 39) das oben Beschriebene erstellt.

Listing 39: Sniff-Log HTML

```
<div class="sniff-log" @ref="sniffLogContainer">
    @if (!SniffLogService.sniffIsPlanned)
    {
        @foreach (var packet in packets)
        {
            if (DateTime.ParseExact(packet.TimeReceived, "yyyy-MM-dd-HH~mm~ss.fffffff",
                System.Globalization.CultureInfo.InvariantCulture).CompareTo(SniffLogService.
                LastClearedTimestamp) > 0)
            {
                <div class="packet">
                    <pre class="packet-text">@packet.ToString()</pre>
                </div>
            }
        }
    }
    else
    {
        <h3>Sniff is planned</h3>
        <p>Start: @SniffLogService.startTime</p>
        <p>End: @SniffLogService.endTime</p>
    }
</div>
```

Durch eine Abfrage, ob eine Aufzeichnung geplant ist, wird entschieden, ob der Nutzer nun die Pakete oder die Informationen zur geplanten Aufzeichnung sieht. Falls die Pakete angezeigt werden sollen, wird für jedes im Sniff-Log gespeicherte Paket, das nach dem gespeicherten Zeitpunkt erstellt wurde, ein HTML-Element erstellt. Falls die Informationen der geplanten Aufzeichnung angezeigt werden sollen, wird eine Überschrift angezeigt, in der steht, dass eine Aufzeichnung geplant ist. Dazu werden noch der Start- und Endzeitpunkt als normale Texte angezeigt.

Bei der Initialisierung wird wieder ein Timer gestartet (siehe Listing 40).

Listing 40: Sniff-Log Timer

```
protected override async Task OnInitializedAsync()
{
    //Timer that start when SniffLog.razor is initialized in Home.razor and is called
        every second.
    //The methods LoadPackets() and UpdateStatus() are called asynchronous.
    //The method ScrollToBottom() is called asynchronous and only if
        "SniffLogService.isRunning" or "scrolling" are set true.
    //In the end the method StateHasChanged() is called to tell the component that there
        have been changes.
    timer = new Timer(async _ =>
    {
        await LoadPackets();
        await UpdateStatus();
        if (SniffLogService.isRunning || scrolling)
        {
            await ScrollToBottom();
        }
        StateHasChanged();

        scrolling = SniffLogService.isRunning;
    }, null, 0, 1000);
}
```

Dieser Timer ruft jede Sekunde die Methoden LoadPackets(), UpdateStatus() und StateHasChanged() auf. Die Methode ScrollToBottom wird nur aufgerufen, wenn im Moment eine Aufzeichnung läuft oder der Boolean „scrolling“ „true“ ist. „scrolling“ übernimmt am Ende des Timers den Wert von „SniffLogService.isRunning()“. Dies wird gemacht, damit solange eine Aufzeichnung läuft und noch einmal danach nach unten gescrollt wird. Diese Funktionalität wurde mittels einer JavaScript-Funktion umgesetzt (siehe Listing 41).

Listing 41: Scroll To Bottom JavaScript-Funktion

```
<script>
    window.scrollToBottom = (element) => {
        element.scrollTop = element.scrollHeight - element.clientHeight;
    };
</script>
```

Dabei wird sich ausgerechnet, welcher der oberste Punkt ist, der angezeigt wird, damit das unterste Paket ganz unten im Sniff-Log angezeigt wird. Dieser Wert wird in die Eigenschaft „scrollTop“ eingesetzt.

Listing 42: Scroll To Bottom C#-Implementierung

```
private async Task ScrollToBottom()
{
    await JSRuntime.InvokeVoidAsync("scrollToBottom", sniffLogContainer);
}
```

In der Methode ScrollToBottom() (siehe Listing 42) wird JSRuntime verwendet, um den JavaScript-Code auszuführen. Dabei wird die „scrollToBottom“-Methode auf das sniffLogContainer-Element ausgeführt.

Beim Stoppen der Aufzeichnung kommuniziert der Sniff-Log mit dem API-Service. In der Methode StopRecording() wird das API-Service benutzt, um mit der API zu kommunizieren. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt.

Listing 43: Sniff-Log Pakete laden

```
private async Task LoadPackets()
{
        if (packetArr.Length > 0)
        {
            if (packetArr[0] != null)
            {
                packets = new List<Packet>(packetArr);
                userWasInformed = false;
            }
            else if(!userWasInformed)
            {
                await PopUpMessage("Backend could not be reached!");
                informationPopUp.Show();
                userWasInformed = true;
            }
        }

        StateHasChanged();
}
```

Um die Pakete zu laden, wird die Methode LoadPackets() (siehe Listing 43) erstellt. Die Pakete, die von ApiService.GetPackets()geholt werden, Ausführung der Anfrage beschrieben in ApiService(8.3.6), werden in ein Array gespeichert. Danach folgt eine Abfrage, ob die Länge des Arrays größer als 0 ist. Falls dies der Fall ist, folgt noch eine Abfrage, die überprüft, ob der erste Wert „null“ ist. Falls dies der Fall ist, konnte das Backend nicht erreicht werden. In diesem Fall wird ein Informations-Pop-Up (8.3.2) mit der Nachricht „Backend could not be reached!“ angezeigt. Wenn der erste Wert des Arrays nicht „null“ ist, werden die Werte aus dem Array in die Paket-Liste übernommen. In dieser Funktion wird der Boolean „userWasInformed“ verwendet, damit das Pop-Up nicht jede Sekunde wieder geöffnet wird, wenn das Backend nicht erreicht werden konnte. Zum Schluss wird die Methode StateHasChanged() ausgeführt, um die Komponente über die Änderungen zu informieren.

Der Sniff-Log kann mit der Methode ClearPackets() geleert werden. Dabei wird einfach die Paketliste geleert.

Listing 44: Sniff-Log Status

```
1  private async Task UpdateStatus()
2  {
3      string[] times = await ApiService.Schedule();
4      if (await ApiService.IsRecording())
5      {
6          SniffLogService.isRunning = true;
7          if (times.Length <= 1 || times[0] == null ||
               times[0].CompareTo(DateTime.Now.ToString("dd.MM.yyyy HH:mm")) <= 0)
8          {
9              SniffLogService.sniffIsPlanned = false;
10         }
11         else
12         {
13             SniffLogService.sniffIsPlanned = true;
14             SniffLogService.startTime = times[0];
15             SniffLogService.endTime = times[1];
16         }
17
18     }
19     else
20     {
21         SniffLogService.isRunning = false;
22         SniffLogService.sniffIsPlanned = false;
23     }
24 }
```

Die Methode UpdateStatus() (siehe Listing 44) kümmert sich darum, den Status des Sniff-Logs zu ändern. Die Zeiten, die von ApiService.Schedule() geholt werden, Ausführung der Anfrage beschrieben in ApiService(8.3.6), werden in ein Array gespeichert. Danach wird der Boolean SniffLogService.isRunning auf „true“ gesetzt und es folgt eine weitere Abfrage, mit der abgefragt wird, ob ApiService.isRunning() „true“ ist. Ausführung der Anfrage beschrieben in ApiService(8.3.6). Falls dies der Fall ist, folgt eine weitere Abfrage, mit der überprüft wird, ob die Länge von dem Array kleiner gleich 1 ist, der erste Wert „null“ ist oder der Anfangszeitpunkt der Aufzeichnung in der Vergangenheit liegt und somit die Aufzeichnung schon läuft. Falls dies der Fall ist, ist keine Aufzeichnung geplant und der Wert SniffLogService.sniffIsPlanned wird auf „false“ gesetzt. Falls dies nicht der Fall ist, wird der Wert SniffLogService.sniffIsPlanned auf „true“ gesetzt und der Start- und Endzeitpunkt werden im SniffLogService gespeichert. Wenn die erste Abfrage schon „false“ war, werden die Boolean SniffLogService.isRunning und SniffLogService.sniffIsPlanned auf „false“ gesetzt.

Recordings-Configuration-Pop-Up

Das Recordings-Configuration-Pop-Up, Klasse RecordingsConfigurationPopUp.razor, ist ein Pop-Up, das angezeigt wird, wenn der Nutzer auf den Start-Button auf der Hauptseite drückt. Es zeigt die verbundenen Netzwerktypen WLAN und LAN an. Um eine Aufzeichnung zu starten, muss mindestens einer davon ausgewählt werden, es können aber auch beide Netzwerktypen ausgewählt werden. Weiterhin bietet es die Möglichkeit, eine Aufzeichnung zu planen, indem eine Zeitspanne ausgewählt werden kann. Dies ist aber optional. Falls keine Verbindung zu einem Netzwerk besteht oder diese noch geladen werden, steht statt der Netzwerke der Text „Loading networks...“ geschrieben.

Das Recordings-Configuration-Pop-Up hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout identisch.

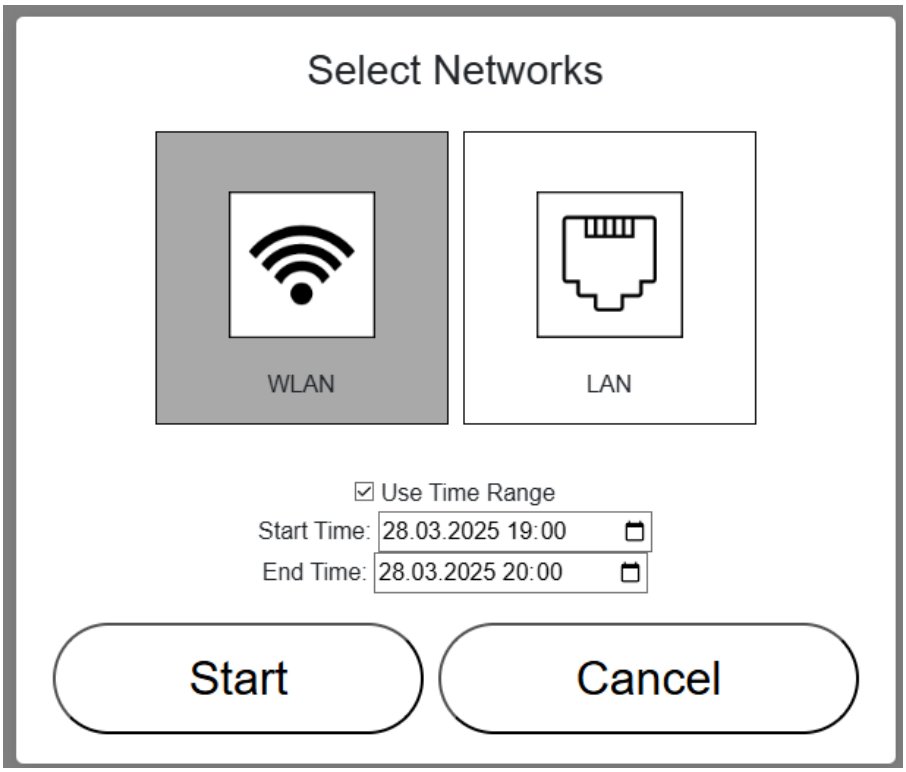

Abbildung 32: Recordings-Configuration-Pop-Up

Umsetzung: Zuerst werden Imports und Injects gemacht, diese erfolgen wie im Homescreen. Die Implementierung der Sichtbarkeit funktioniert wie bei dem Informations-Pop-Up.

Damit das Pop-Up angezeigt werden kann, beinhaltet es die Methode ShowPopup() (siehe Listing 45).

Listing 45: ShowPopUp() Methode

```
1  public async Task ShowPopup(string filter)
2  {
3      filterString = filter;
4      try
5      {
6          string[] networks = await ApiService.GetConnections() ?? Array.Empty<string>();
7
8          connectedNetworks.Clear();
9          foreach (var network in networks)
10         {
11             connectedNetworks.Add(new Network { SSID = network.Replace("wifi",
                   "WLAN").Replace("br0", "LAN").Split(":")[0] });
12         }
13
14         SetDefaultTimes();
15         useTimeRange = false;
16
17         showPopup = true;
18         StateHasChanged();
19     }
20     catch (Exception ex)
21     {
22         Console.WriteLine($"Error loading networks: {ex.Message}");
23     }
24 }
```

Dabei wird beim Aufrufen der Methode ein String mitgegeben, der die Filter für die Aufzeichnung enthält. Am Anfang der Methode wird dieser String vom String „filterString“ übernommen. Danach wird ein String-Array „networks“ erstellt. Dieses String-Array übernimmt die Rückgabe von ApiService.GetConnections(). Falls dabei „null“ zurückgegeben wird, wird es stattdessen zu einem leeren Array. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Anschließend wird die Liste der Netzwerke mit den Werten des gerade erstellten Strings befüllt. Dabei werden die Werte abgeändert, damit besser damit gearbeitet werden kann. Im nächsten Schritt wird die Methode SetDefaultTimes() aufgerufen, die gleich noch erklärt wird und „useTimeRange“ wird auf „false“ gesetzt.

Zum Schluss wird dann der Wert „shopPopup“ auf „true“ gesetzt, damit das Pop-Up angezeigt wird und StateHasChanged() aufgerufen wird, um die Komponente über die Änderungen zu informieren.

Für die Implementierung des Anzeigens der Netzwerke im Pop-Up wird im HTML Folgendes implementiert:

Listing 46: Netzwerke im Recordings-Configuration-Pop-Up

```
@if (connectedNetworks == null || !connectedNetworks.Any())
{
    <p>Loading networks...</p>
}
else
{
    @foreach (var network in connectedNetworks)
    {
        <div class="@GetNetworkClass(network)" @onclick="() =>
            ToggleNetworkSelection(network)">
            <img src="images/@GetNetworkIcon(network)" class="icon">
            <label>@network.SSID</label>
        </div>
    }
}
```

Dabei wird überprüft, ob entweder die Liste der verbundenen Netzwerke „null“ ist oder die Liste leer ist. Falls dies der Fall ist, wird der Text „Loading networks...“ angezeigt. Falls dies nicht der Fall ist, werden die verbundenen Netzwerke mit einem Bild angezeigt. Dieses Bild wird über die Methode GetNetworkIcon() bestimmt. Dabei schickt die Methode den String mit dem Namen des Bildes zurück. „wifi.png“ falls die SSID des Netzwerks „WLAN„ beinhaltet und „network.png“ falls nicht. Die Implementierung der GetNetworkClass funktioniert gleich, mit dem einzigen Unterschied, dass ein Boolean abgefragt wird und die Rückgabewerte „selectedNetwork“ und „network“ sind. Um den Wert von IsChecked eines Netzwerks zu ändern, kann auf diese gedrückt werden. Dabei wird die Methode ToggleNetworkSelection() aufgerufen, die den Wert von IsChecked umkehrt. Für die Implementierung der Zeitspannen-Auswahl wird im HTML Folgendes implementiert:

Listing 47: Zeitspanne im Recordings-Configuration-Pop-Up

```
<div>
    <input type="checkbox" @bind="UseTimeRange" /> Use Time Range
</div>
@if (useTimeRange)
{
    <div>
        <label>Start Time:</label>
        <input type="datetime-local" @bind="startTime" min="@GetStartTime()" />
    </div>
    <div>
        <label>End Time:</label>
        <input type="datetime-local" @bind="endTime" min="@GetEndTime()" />
    </div>
}
```

Dabei wird eine Checkbox erstellt, diese ist mit dem Property UseTimeRange (siehe Listing 48 verbunden.

Listing 48: Zeitspannen Property

```
private bool UseTimeRange
{
    get => useTimeRange;
    set
    {
        useTimeRange = value;
        if (!useTimeRange)
        {
            SetDefaultTimes();
        }
    }
}
```

Dabei wird der Wert von useTimeRange von UseTimeRange übernommen. Falls dieser Wert dann „false“ ist, wird die Methode SetDefaultTimes() (siehe Listing 49) aufgerufen.

Listing 49: SetDefaultTimes() Methode

```
private void SetDefaultTimes()
{
    var now = DateTime.Now;

    int minutesToAdd = (30 - (now.Minute % 30)) % 30;
    if (minutesToAdd == 0) minutesToAdd = 30;

    startTime = now.AddMinutes(minutesToAdd).Date + new TimeSpan(now.Hour,
        now.Minute+minutesToAdd, 0);

    endTime = startTime.AddHours(1);
}
```

Dabei werden für den Start- und Endzeitpunkt angemessene Zeitpunkte in der Zukunft gewählt. Es wird die nächste halbe Stunde berechnet. Dieser Wert wird dem Startzeitpunkt zugeteilt. Der Endzeitpunkt wird auf eine Stunde nach dem Startzeitpunkt gesetzt.
Danach folgt eine Abfrage, die überprüft, ob nun eine Zeitspanne verwendet wird. Falls dies der Fall ist, kann der Nutzer einen Start- und Endzeitpunkt eingeben. Die Methoden GetStartTime() und GetEndTime() definieren den frühesten Wert, der in den Feldern stehen darf.

Am Ende des HTMLs werden noch zwei Buttons definiert (siehe Listing 50).

Listing 50: Buttons im Recording-Configuration-Pop-Up

```
<button class="button" @onclick="ConfirmStart" disabled="@(!CanStart())">Start</button>
<button class="button" @onclick="Cancel">Cancel</button>
```

Dabei gibt es einen Cancel-Button, die verbundene Methode Cancel() ist genauso implementiert, wie die Close() Methode des Informations-Pop-Ups. Der Start-Button ist deaktiviert, solange die Methode CanStart() „false“ zurückgibt. Die Methode CanStart() gibt „true“ zurück, wenn mindestens ein Netzwerk ausgewählt ist und entweder keine Zeitspanne verwendet wird oder der

Startzeitpunkt nach dem gegenwärtigen Moment und der Endzeitpunkt nach dem Startzeitpunkt liegt.

Die Methode ConfirmStart() (siehe Listing 51) erstellt einen Body, der für die API-Anfrage verwendet wird. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Die Namen der ausgewählten Netzwerke werden in ein String-Array gespeichert. Im nächsten Schritt wird eine Liste aus Strings erstellt. Der erste String ist der Filter-String, der zweite String setzt sich aus den ausgewählten Netzwerken zusammen. Danach wird abgefragt, ob eine Zeitspanne verwendet wird. Falls dies der Fall ist, wird der Startzeitpunkt als dritter String mitgegeben und der Endzeitpunkt als vierter String. Zum Schluss wird ApiService.StartRecording() mit dem Body als Array aufgerufen und das Pop-Up geschlossen.

Listing 51: ConfirmStart() Methode

```
1  private async Task ConfirmStart()
2  {
3      //The names of the selected networks are saved into "selectedNetworks"
4      var selectedNetworks = connectedNetworks.Where(n => n.IsChecked).Select(n =>
           n.SSID).ToArray();
5      //body is being created as List<string> and filterString and selectedNetworks are
           saved as strings into the body.
6      var body = new List<string>
7      {
8          filterString,
9          string.Join(",", selectedNetworks),
10     };
11     body[1] = body[1].Replace("WLAN", "wlan0").Replace("LAN", "br0");
12     //Checks if useTimeRange is true.
13     //true: adds the times to the body and sets SniffLogService.sniffIsPlanned to true.
14     //false: Nothing happens.
15     if (useTimeRange)
16     {
17         body.Add(startTime.ToString("o"));
18         body.Add(endTime.ToString("o"));
19     }
20
21     //Calls the method ApiService.StartRecording() with body as an array and sets
           showPopup to false.
22     ApiService.StartRecording(body.ToArray());
23     showPopup = false;
24 }
```

8.3.4 Recordings

Page

Die Recordings-Seite, Klasse RecordingsMenu.razor, ist die Seite der RaspiSniffer-Weboberfläche, in der die gespeicherten Aufzeichnungen angezeigt werden. Diese können dann entweder auf einen USB-Stick oder ein anderes Gerät geladen oder gelöscht werden. Die Seite beinhaltet eine Box, den File-Container, der die gespeicherten Dateien aus dem Backend anzeigt. Die Recordings-Seite beinhaltet fünf Buttons:

- Back: Der Back-Button führt den Benutzer zurück zur Hauptseite.
- Delete: Der Delete-Button schickt die Namen der ausgewählten Dateien an das Backend. Dieses löscht dann die ausgewählten Dateien.
- To USB: Der To-USB-Button öffnet das USB-Selection-Pop-Up, in dem ein USB-Stick ausgewählt und ein Speicherpfad eingegeben werden kann. Dieses kann nur geöffnet werden, falls eine oder mehrere Dateien ausgewählt sind.
- Dismount USB: Der Dismount-USB-Button öffnet das USB-Dismount-Pop-Up, in dem ein USB-Stick ausgewählt werden kann.
- Download: Der Recordings-Button führt den Benutzer auf die Recordings-Seite, auf der der Benutzer Files auf das eigene Gerät herunterladen, auf einen USB-Stick kopieren oder löschen kann. Der Download-Button lässt den Benutzer die ausgewählten Dateien auf das Gerät herunterladen. Dieser Button ist nur im 16:9-Layout sichtbar, da keine Datei auf dem RaspiSniffer heruntergeladen werden muss.

Die Recordings-Seite hat zwei verschiedene Layouts, eine für einen 4:3-Bildschirm, Raspberry Pi-Bildschirm, und eine für einen 16:9-Bildschirm, PC-Bildschirm:

16:9-Layout: Beim 16:9-Layout sind die Buttons am unteren Rand platziert und der File-Container belegt die oberen 75 Prozent der Seite. Im Gegensatz zum 4:3-Layout befindet sich inmitten der Buttons der Download-Button.

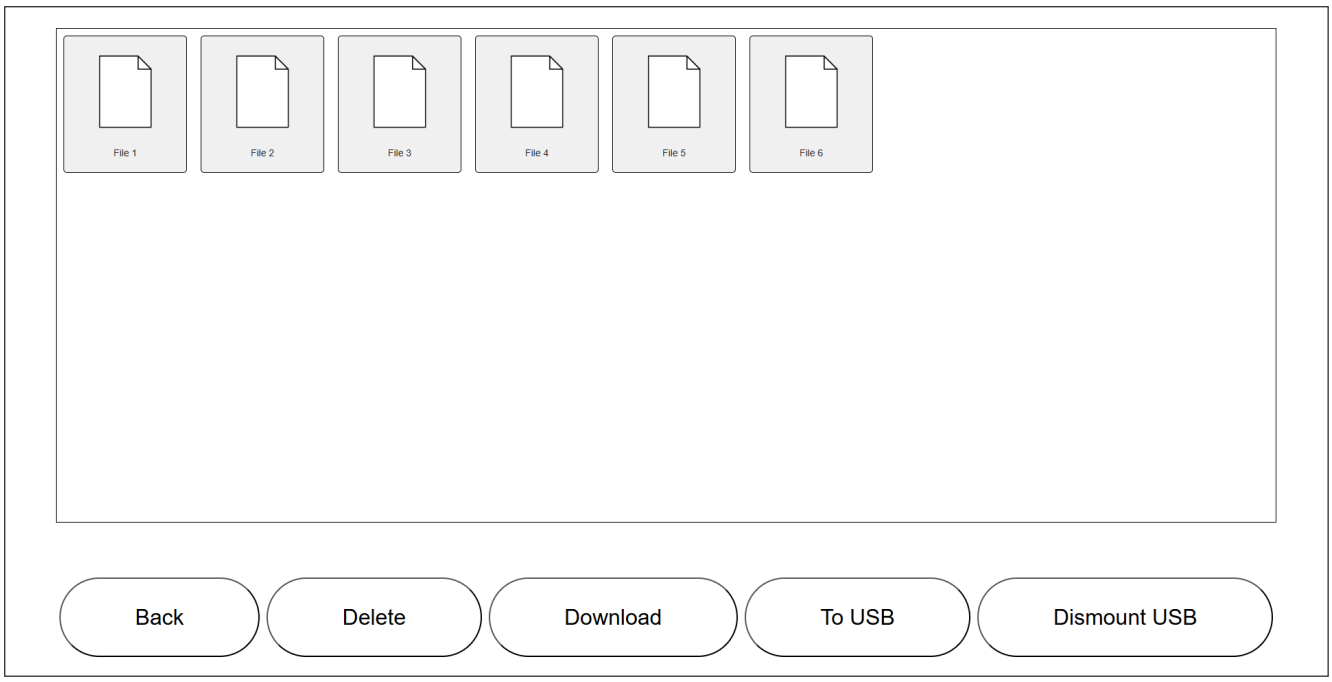

Abbildung 33: 16:9-Bildschirm Recordings-Seite

4:3-Layout: Beim 4:3-Layout sind die Buttons am unteren Rand platziert und der File-Container belegt die oberen 80 Prozent der Seite. Im Gegensatz zum 16:9-Layout befindet sich inmitten der Buttons kein Download-Button.

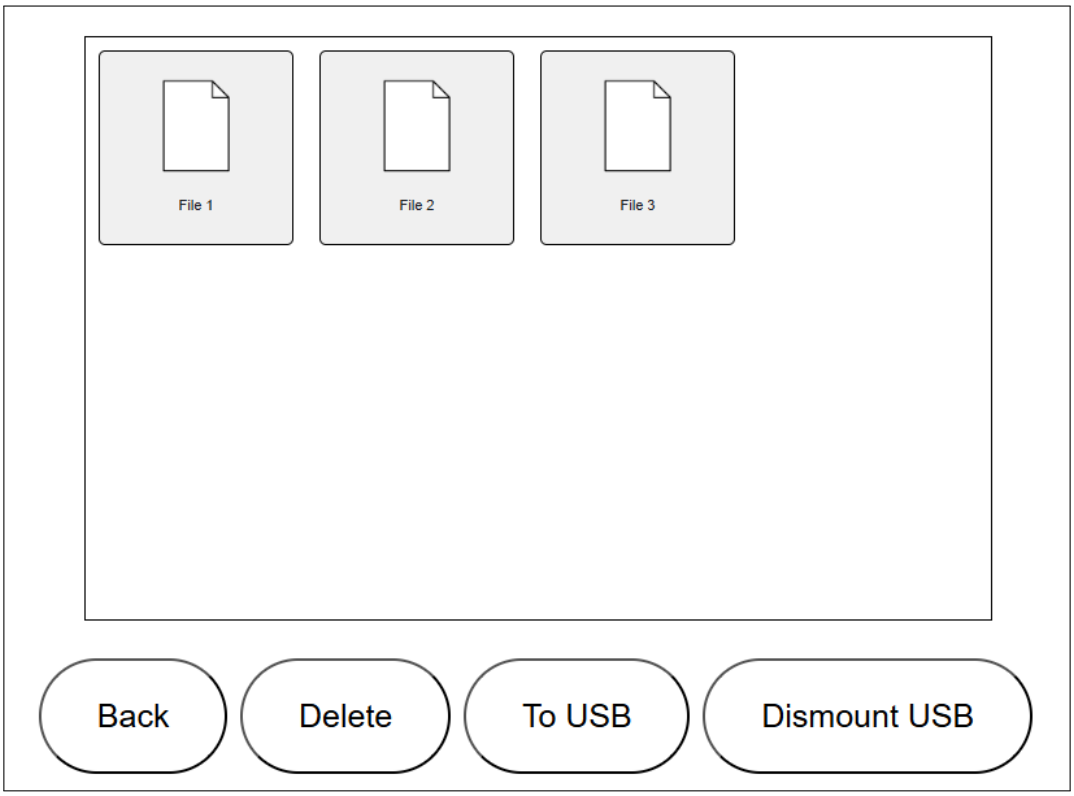

Abbildung 34: 4:3-Bildschirm Recordings-Seite

Umsetzung: Die Implementierung der URL, der Imports und Injects, des Seitentitels sowie der Komponenten erfolgt wie beim Homescreen. Die Komponenten, die verwendet werden, sind der File-Container, das Informations-Pop-Up, das USB-Dismount-Pop-Up, das USB-Selection-Pop-Up und das Cancellation-Pop-Up.

Der Back-Button hat die gleiche Implementierung wie der WIFI-Button im Homescreen. Mit dem Unterschied, dass dieser zu „/“ weiterleitet.

Beim Drücken des Delete-Buttons, des Download-Buttons und des To-USB-Buttons wird abgefragt, ob eine oder mehrere Dateien ausgewählt sind. Falls dies nicht der Fall ist, wird ein Informations-Pop-Up (8.3.2) mit dem Text „Please select a recording“angezeigt. Falls nun eine oder mehrere Dateien ausgewählt sind, führen die Buttons zu ihren jeweiligen Methoden.

Beim Delete-Button wird die Methode DeleteFile() aufgerufen, diese benutzt die im File-Container implementierte Methode DeleteSelectedFiles().

Beim Download-Button wird die Methode DownloadFiles() (siehe Listing 53) aufgerufen. Dies funktioniert über die JavaScript-Funktion downloadFile (siehe Listing 52). Dabei wird ein unsichtbares Element erstellt, das die URL öffnet, die zum Download führt. Dieses schließt sich dann wieder selbst. Im C#-Code wird für jede ausgewählte Datei diese URL erstellt und mit JSRuntime wird die JavaScript-Funktion dann ausgeführt. Das Herunterladen der Dateien wird dem Browser überlassen. Somit auch die damit zusammenhängende Auswahl des Speicherpfads und die Benennung.

Listing 52: Download JavaScript-Funktion

```
<script>
    window.downloadFile = (url) => {
        var a = document.createElement("a");
        a.href = url;
        a.download = url.split('/').pop();
        a.style.display = "none";
        document.body.appendChild(a);
        a.click();
        document.body.removeChild(a);
    }
</script>
```

Listing 53: Download C#-Implementierung

```
private async Task DownloadFiles()
{
    var selectedFiles = fileContainer.GetSelectedFiles();

    foreach (var file in selectedFiles)
    {
        var filePath = $"/recordings/{file}";
        await JSRuntime.InvokeVoidAsync("downloadFile", filePath);
    }
}
```

Im To-USB-Button und dem Dismount-USB-Button wird die Methode LoadUsb() aufgerufen, diese lädt die an den RaspiSniffer angesteckten USB-Sticks und speichert diese in die USB-Liste. Danach wird das jeweilige Pop-Up geöffnet.

Bei der Methode HandleDismount() wird ein Body erstellt, in dem die ausgewählten USB-Stick als String-Array gespeichert werden und dann eine Anfrage an die API mit dem erstellten Body geschickt wird. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt.

Bei der Methode HandleUSBConfirm() (siehe Listing 54) wird der ausgewählte USB-Stick und Pfad in ein Array zusammengefasst. Danach öffnet sich das Cancellation-Pop-Up (8.3.2) welches solange geöffnet bleibt, bis der Kopiervorgang entweder abgeschlossen oder abgebrochen worden ist. Im Anschluss wird ein Informations-Pop-Up (8.3.2) angezeigt, welches den Ausgang des Prozesses ausgibt. Bei der Methode HandleCancelUsb() wird eine Anfrage an die API geschickt, die den Kopiervorgang abbricht. Das Cancellation-Pop-Up wird im Anschluss geschlossen.

Listing 54: HandleUSBConfirm() Methode

```
1  private async void HandleUSBConfirm()
2  {
3      var usb = usbSelectionPopUp.GetUSB();
4      var path = usbSelectionPopUp.GetPath();
5      var selectedFiles = new[]
6      {
7          usb, path
8      };
9
10     cancellationPopUp.Show();
11
12     var response = await
           ApiService.ToUsb(selectedFiles.Concat(fileContainer.GetSelectedFiles()).ToArray());
13
14     cancellationPopUp.Close();
15
16     informationPopUp.Message = $"Status code: {response.StatusCode}";
17     informationPopUp.Show();
18 }
```

File-Container

Der File-Container, Klasse FileContainer.razor, ist die Komponente, die die Aufzeichnungen im Frontend visuell darstellt. Dieser zeigt die Namen der Dateien an. Diese werden im Frontend nicht sortiert und werden in der Reihenfolge angezeigt, in der die API die Dateien schickt.

Die Dateien werden in einer kleinen Box, mit einem generischen Bild und dem Namen angezeigt. Das Format für den Namen der Datei ist „„IP-Adresse des RaspiSniffers“_HH ˜ mm ˜ ss_dd-MM.pcap“. Zur besseren visuellen Erkennung können diese Boxen in vier Graustufen unterschieden werden. Diese sind:

- Hellgrau: Die Datei ist nicht ausgewählt.
- Grau: Die Datei ist nicht ausgewählt, der Mauszeiger liegt auf der Box.

- Dunkelgrau: Die Datei ist ausgewählt.
- Anthrazit: Die Datei ist ausgewählt, der Mauszeiger liegt auf der Box.

Der Benutzer hat die Möglichkeit, keine, eine oder mehrere Dateien auszuwählen.

Das 16:9-Layout und das 4:3-Layout unterscheiden sich nur minimal. Dabei wird nur die Größe der Boxen und der Inhalt dieser geändert.

Abbildung 35: 16:9-Bildschirm File-Container

Abbildung 36: 4:3-Bildschirm File-Container

Umsetzung:

Zuerst werden Imports und Injects gemacht, diese erfolgen wie im Homescreen. Um die Recordings anzuzeigen, wird im HTML (siehe Listing 55) das oben Beschriebene erstellt.

Listing 55: File-Container HTML

```
<div class="files" @ref="file_Container">
    @foreach (var file in files)
    {
        <div class=@(IsSelected(file) ? "selectedFile" : "file") @onclick="() =>
            ToggleSelection(file)">
            <label class="file-label" data-file="@file">
                <img src="@imageSource" class="icon">
                <div class="file-text">@file</div>
            </label>
        </div>
    }
</div>
```

Für jede Datei in der Dateien-Liste wird eine Box angelegt. Diese wird mit einem generischen Bild und dem Namen angezeigt. Die Klasse der Box wird durch die Methode IsSelected() bei jeder Datei abgefragt. In dieser Methode wird einfach abgefragt, ob die Datei in der Liste der ausgewählten Dateien enthalten ist. Beim Drücken auf eine Box wird die Methode ToggleSelection() aufgerufen, in dieser wird abgefragt, ob die Datei in der Liste der ausgewählten Dateien enthalten ist. Falls dies der Fall ist, wird die Datei entfernt und falls dies nicht der Fall ist, wird die Datei in die Liste hinzugefügt. Um die Dateien aktuell zu halten, wird in der Methode OnInitializedAsync() ein Timer gestartet. Dieser Timer führt jede Sekunde die Methode LoadFiles() (siehe Listing 56) aus. Diese funktioniert genauso wie die LoadPackets() Methode im SniffLog. Mit dem Unterschied, dass anstatt der Pakete die Dateien von der API geholt werden.

Listing 56: LoadFiles() Methode

```
private async Task LoadFiles()
{
    string[] recordings = await ApiService.GetRecordings();
    if (recordings.Length > 0)
    {
        informationPopUp.Show() method is called.
        if (recordings[0].CompareTo(string.Empty) != 0)
        {
            files = new List<string>(recordings);
            userWasInformed = false;
        }
        else if (!userWasInformed)
        {
            await PopUpMessage("Backend could not be reached!");
            informationPopUp.Show();
            userWasInformed = true;
        }
    }
}
```

Weiter bietet der File-Container die Möglichkeit zum Löschen von Dateien. Dies funktioniert über die Methode DeleteSelectedFiles(). Diese schickt ein Array der ausgewählten Dateien an die API, mittels der Methode DeleteRecording() im ApiService. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Danach wird die Methode LoadFiles() aufgerufen, damit die gelöschten Dateien nicht mehr im File-Container angezeigt werden. Der File-Container bietet auch die Möglichkeit, mittels der Methode HasSelectedFiles(), abzufragen, ob es ausgewählte

Dateien gibt. Dazu bietet er auch die Möglichkeit, die ausgewählten Dateien abzufragen, mittels der Methode GetSelectedFiles().

USB-Selection-Pop-Up

Das USB-Selection-Pop-Up, Klasse USBSelectionPopUp, ist ein Pop-Up, das angezeigt wird, wenn der Nutzer auf den To-USB-Button auf der Recordings-Seite drückt. Dabei muss mindestens eine Datei ausgewählt sein. Es zeigt die mit dem RaspiSniffer verbundenen USB-Sticks an. Um den Kopiervorgang zu starten, muss einer davon ausgewählt werden, dabei kann nur ein USB-Stick ausgewählt sein. Weiterhin bietet es die Möglichkeit, einen Speicherpfad auf dem USB-Stick einzugeben, dieser ist automatisch „/Recordings/“. Falls kein USB-Stick angeschlossen ist oder erkannt wird, steht statt der USB-Sticks der Text „No USB detected!“ geschrieben.

Solange das Pop-Up geöffnet ist, ist der Hintergrund grau.

Das USB-Selection-Pop-Up hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout identisch.

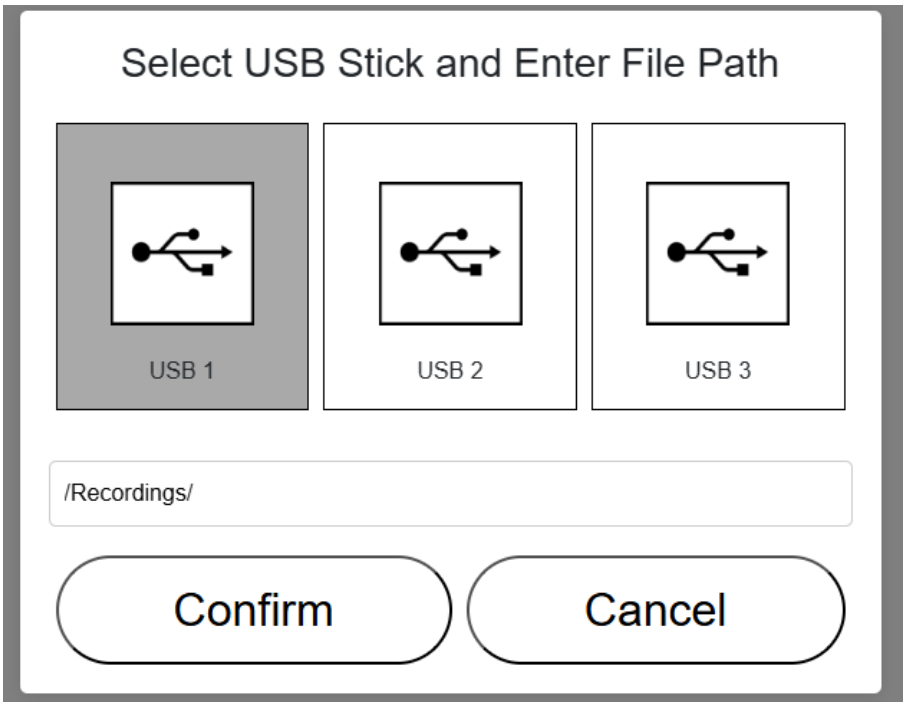

Abbildung 37: USB-Selection-Pop-Up

Umsetzung: Die Implementierung der Sichtbarkeit ist wie bei Informations-Pop-Up. Damit das Pop-Up angezeigt werden kann, beinhaltet es die Methode Show(). Diese funktioniert ähnlich wie die Implementierung dieser Methode im Informations-Pop-Up, mit dem Unterschied, dass die Variable, die den Pfad speichert, auf „/Recodings/“ gesetzt wird. Für die Implementierung des Anzeigens der USB-Sticks im Pop-Up wird im HTML Folgendes implementiert:

Listing 57: USB-Selection-Pop-Up USB anzeigen

```
@if (USBSticks.Count > 0)
{
    @foreach (var usb in USBSticks)
    {
        <div class=@(isSelected(usb) ? "selectedUSB" : "usb") @onclick="() =>
            ToggleSelection(usb)">
            <img src="@imageSource" class="icon"> <!-- Image tag for the USB icon -->
            <label>@usb</label>
        </div>
    }
}
else
{
    <h4 class="warning-text">No USB detected!</h4>
}
```

Dabei wird überprüft, ob USB-Sticks in der USB-Liste gespeichert sind. Falls dies der Fall ist, wird für jeden USB-Stick eine Box angezeigt. In dieser Box wird ein generisches Bild mit dem Namen eines USB-Sticks angezeigt. Der ausgewählte USB-Stick wird in einem String gespeichert. Bei ToggleSelection() wird bei einem String abgefragt, ob dies der Name des ausgewählten USB-Sticks ist. Falls dies der Fall ist, wird der String auf „null“ gesetzt. Falls dies nicht der Fall ist, wird der String mit dem Namen des USB-Sticks überschrieben. Die Methode isSelected() gibt den Namen des ausgewählten USB-Sticks zurück. Für das Eingeben des Speicherpfads gibt es eine Textbox, die mit dem String verbunden ist, der den Speicherpfad speichert. Am Ende des HTMLs werden noch zwei Buttons definiert (siehe Listing 58).

Listing 58: Buttons im USB-Selection-Pop-Up

```
<button class="button" @onclick="Confirm" disabled="@(IsConfirmEnabled)">Confirm</button>
<button class="button" @onclick="Cancel">Cancel</button>
```

Dabei gibt es einen Cancel-Button, die verbundene Methode Cancel() setzt die Werte des ausgewählten USB-Sticks und des Speicherpfads auf die Standardwerte zurück. Danach ruft sie die Methode Close() auf, die genauso wie die Methode Close() im Information-Pop-Up implementiert ist. Es gibt den Confirm-Button, dieser ist deaktiviert, solange „IsConfirmEnabled“ „true“ zurückgibt. Das macht er, solange kein USB-Stick ausgewählt ist und solange der Speicherpfad ungültig ist. Wenn der Confirm-Button gedrückt wird, wird die Methode Confirm() ausgeführt. Diese fragt ab, ob am Anfang und Ende des Speicherpfades ein „/“ steht. Falls dies nicht der Fall ist, wird an jeweiliger Position eines hinzugefügt. Danach wird der EventCallback OnConfirm

ausgeführt, der die in der Erstellung der Komponente angegebene Methode ausführt. Zum Schluss der Methode wird die Methode Close() ausgeführt.

Weiter bietet das Pop-Up mit GetUSB() die Möglichkeit, den ausgewählten USB-Stick abzufragen und mit GetPath() den eingegebenen Speicherpfad abzufragen.

USB-Dismount-Pop-Up

Das USB-Dismount-Pop-Up, Klasse USBDismountPopUp, ist ein Pop-Up, das angezeigt wird, wenn der Nutzer auf den Dismount-USB-Button auf der Recordings-Seite drückt. Es zeigt die mit dem RaspiSniffer verbundenen USB-Sticks an. Um den Trennvorgang zu starten, muss mindestens einer davon ausgewählt werden, es können aber auch mehrere USB-Sticks ausgewählt werden. Falls kein USB-Stick angeschlossen ist oder erkannt wird, steht statt der USB-Sticks der Text „No USB detected!“ geschrieben.

Solange das Pop-Up geöffnet ist, ist der Hintergrund grau.

Das USB-Dismount-Pop-Up hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout identisch.

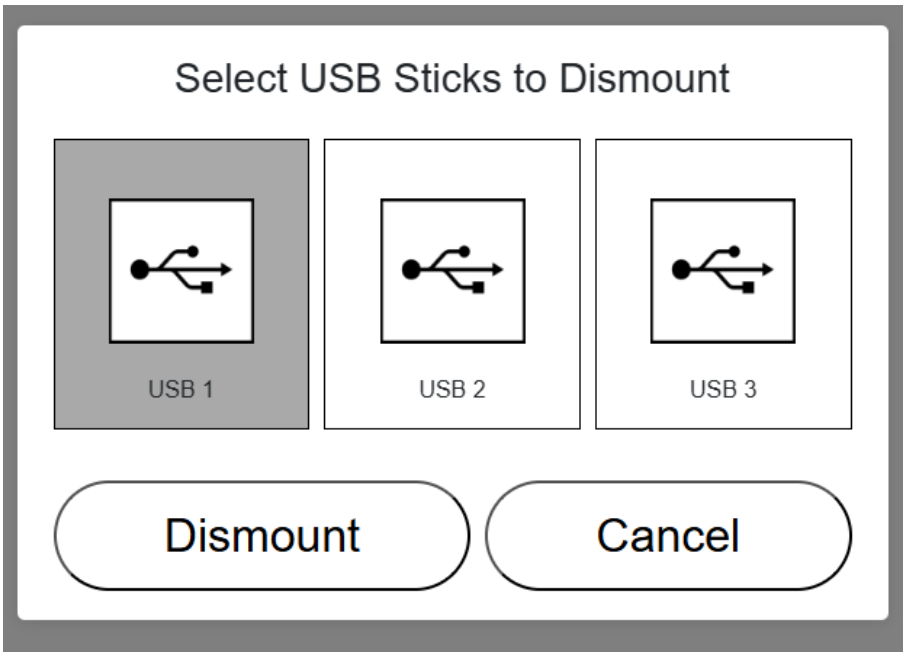

Abbildung 38: USB-Dismount-Pop-Up

Umsetzung: Die Implementierung der Sichtbarkeit ist wie bei Informations-Pop-Up. Damit das Pop-Up angezeigt werden kann, beinhaltet es die Methode Show(). Diese funktioniert genauso wie die Implementierung dieser Methode im Informations-Pop-Up. Die Implementierung des Anzeigens der USB-Sticks ist genauso wie im USB-Selection-Pop-Up.

Am Ende des HTMLs werden noch zwei Buttons definiert (siehe Listing 59)

Listing 59: Buttons im USB-Dismount-Pop-Up

```
1 <button class="button" @onclick="Confirm" disabled="@(IsConfirmEnabled)">Confirm</button>
2 <button class="button" @onclick="Cancel">Cancel</button>
```

Dabei gibt es einen Cancel-Button, der genauso funktioniert wie der Cancel-Button im USB-Selection-Pop-Up. Wobei in diesem Pop-Up nur die Liste der ausgewählten USB-Sticks geleert wird. Es gibt den Dismount-Button, dieser ist deaktiviert, solange die Methode HasSelected() „false“ zurückgibt. Die Methode fragt ab, ob in der Liste der ausgewählten USB-Sticks existieren. Wenn der Dismount-Button gedrückt wird, wird die Methode Dismount() ausgeführt. Dieser führt den EventCallback OnDismount aus, der die in der Erstellung der Komponente angegebene Methode ausführt. Dabei wird die Liste der ausgewählten USB-Sticks an die Methode übergeben. Zum Schluss der Methode wird die Methode Close() ausgeführt.

8.3.5 WIFI

Page

Die WIFI-Seite, Klasse WIFIMenu.razor, ist die Seite der RaspiSniffer-Weboberfläche, in der der Benutzer die Möglichkeit hat, die WLAN-Verbindung zu verwalten. Die Seite beinhaltet zwei Boxen:

- WIFI-Container: Zeigt die möglichen WLAN-Verbindungen an.
- WIFI-Information-Container: Zeigt Informationen und Antworten, die die Backend-API an das Frontend zurücksendet an.

Weiter beinhaltet die WIFI-Seite drei Buttons:

- Back: Der Back-Button führt den Benutzer zurück zur Hauptseite.
- Connect: Der Connect-Button öffnet das Connection-Pop-Up, in dem ein Verschlüsselungprotokoll ausgewählt werden kann. Je nach Verschlüsselungsprotokoll, muss dann eine SSID, ein Passwort und ein Benutzername eingegeben werden. Falls vor dem drücken des Knopfs ein WLAN ausgewählt ist, werden SSID und das Verschlüsselungsprotokoll automatisch ausgefüllt.

- Disconnect: Der Disconnect-Button schickt einen API-Call an das Backend, damit die derzeitige WLAN-Verbindung getrennt wird.

Um sich zu einem anderen WLAN verbinden zu können, darf keine andere WLAN-Verbindung bestehen.

Das WIFI-Seite hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout vom Aufbau identisch. Dafür ist das 4:3-Layout um einiges kompakter und es ändern sich nur ein paar Parameter wie z.B. die Schriftgröße.

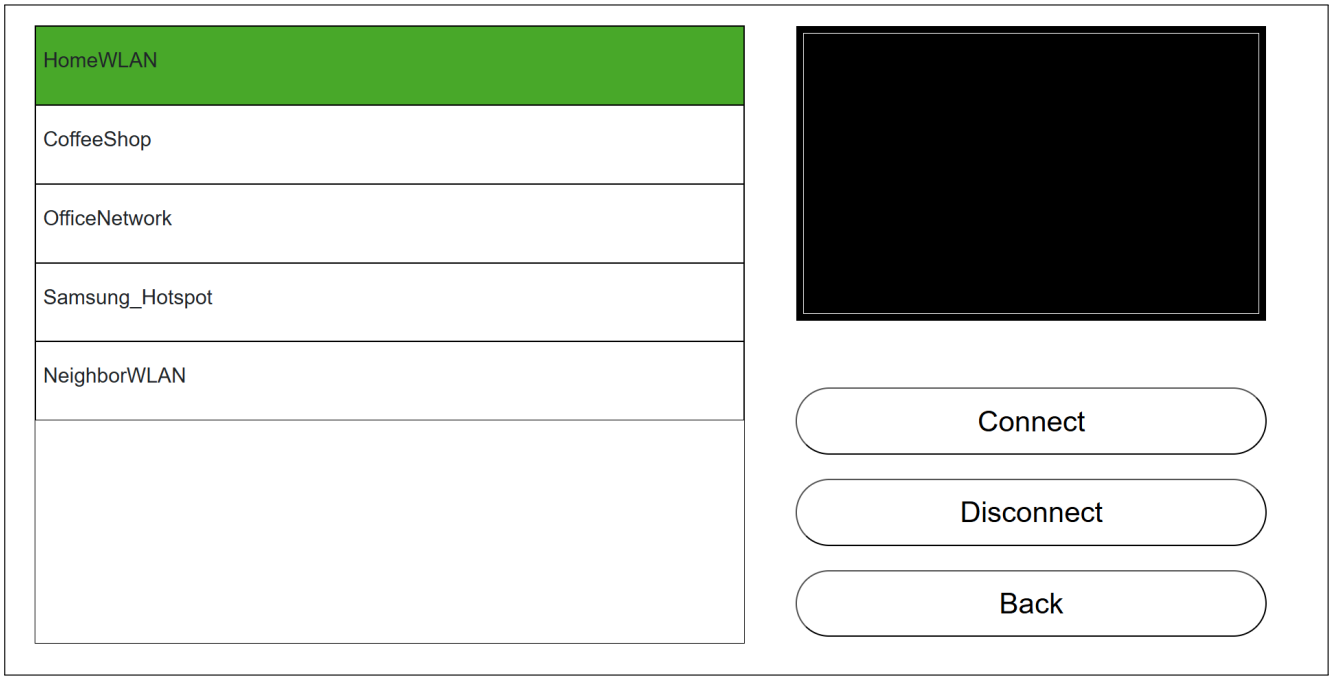

Abbildung 39: 16:9-Bildschirm WIFI-Seite

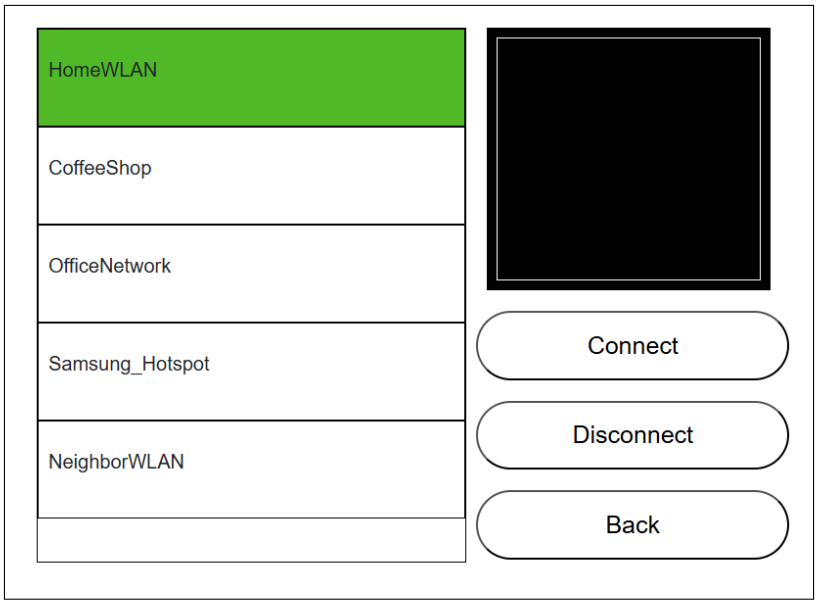

Abbildung 40: 4:3-Bildschirm WIFI-Seite

Umsetzung: Die Implementierung der URL, der Imports und Injects, des Seitentitels sowie der Komponenten erfolgt wie beim Homescreen. Die Komponenten, die verwendet werden, sind der WIFI-Container, der WIFI-Information-Container und das Connection-Pop-Up.

Der Back-Button hat die gleiche Implementierung wie der Back-Button auf der Recordings-Seite.

Der Disconnect-Button führt die Methode TryDisconnect() aus. Diese Methode schreibt „Trying to disconnect...“ in den WIFI-Information-Container, sendet über die Methode DisconnectWifi() im ApiService die Anfrage zum Trennen der WLAN-Verbindung. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Die Antwort wird zwischengespeichert. Die Nachricht wird mitsamt dem Text „Disconnect Status: “ in den WIFI-Information-Container geschrieben.

Der Connect-Button führt die Methode ShowConnectPopUp() (siehe Listing 60). Dabei wird als erstes abgefragt, ob ein WLAN im WIFI-Container ausgewählt ist. Wenn dies nicht der Fall ist, wird das Connection-Pop-Up mit den Standardwerten aufgerufen. Falls dies der Fall ist, folgt eine Abfrage, in der überprüft wird, ob das ausgewählte Netzwerk noch in der Liste der Netzwerke existiert. Falls dies nicht der Fall ist, wird das Connection-Pop-Up mit den Standardwerten aufgerufen. Falls dies der Fall ist, wird die API nach der detaillierten Version des Netzwerks abgefragt. Danach wird der Sicherheitstyp noch etwas bearbeitet, damit im Pop-Up alles richtig eingestellt wird. Zum Schluss wird das Connection-Pop-Up mit der SSID und dem Sicherheitstyp des Netzwerks aufgerufen.

Listing 60: ShowConnectPopUp() Methode

```
private async Task ShowConnectPopUp()
{
    if (wifiContainer.GetSelected() != null)
    {
        var selectedWifi = wifiContainer.GetNetworks().FirstOrDefault(w => w.Ssid ==
            wifiContainer.GetSelected());
        if (selectedWifi != null)
        {
            Wifi wifi = await ApiService.GetWifi(selectedWifi.Ssid);
            if (string.IsNullOrWhiteSpace(wifi.SecurityType) || wifi.SecurityType == "--")
            {
                wifi.SecurityType = "Open";
            }
            if (wifi.SecurityType == "WPA1 WPA2-Enterprise")
            {
                wifi.SecurityType = "WPA2-Enterprise";
            }
            connectionPopUp.Show(wifi.Ssid, wifi.SecurityType);
        }
        else
        {
            connectionPopUp.Show(); // Show pop-up with default values
        }
    }
    else
    {
        connectionPopUp.Show(); // Show pop-up with default values if no Wi-Fi is selected
    }
}
```

Weiter bietet die WIFI-Seite die Methode TryConnect(). Diese Methode schreibt „Attempting to connect to Wi.Fi...“ in den WIFI-Information-Container, sendet über die Methode ConnectWifi() im ApiService die Anfrage zur Herstellung einer WLAN-Verbindung, dabei werden die Verbindungsdetails mitgeschickt. Welche Parameter benutzt werden, wird in der Methode GetConnectionParams() über das Verschlüsselungsprotokoll festgelegt. Die Anfrage wird, wie bei ApiService (8.3.6) beschrieben, ausgeführt. Die Antwort wird zwischengespeichert. Die Nachricht wird mitsamt dem Text „Connect Status: “ in den WIFI-Information-Container geschrieben.

WIFI-Container

Der WIFI-Container, Klasse WIFIContainer.razor, ist die Komponente, die die möglichen WLAN-Verbindungen im Frontend anzeigt. Dieser zeigt den Namen der WLAN-Verbindungen an. Diese werden im Frontend nicht sortiert und werden in der Reihenfolge angezeigt, in der die API die WLAN-Verbindungen schickt. Die einzige Ausnahme ist hierbei das verbundene WLAN, das ganz oben in einer grünen Farbe angezeigt wird.

Die WLAN-Verbindungen werden in einer kleinen Box angezeigt, in der die SSID des WLANs steht. Zur besseren visuellen Erkennung werden die Boxen visuell verändert, falls der Benutzer ein WLAN auswählt oder mit dem Mauszeiger auf einer Box liegt. Diese visuellen Hinweise sind:

- Weißer Hintergrund: Es besteht keine Verbindung zum Netzwerk.
- Grauer Hintergrund: Es besteht keine Verbindung zum Netzwerk, der Mauszeiger liegt auf der Box.
- Grüner Hintergrund: Es besteht eine Verbindung zum Netzwerk.
- Dunkelgrüner Hintergrund: Es besteht eine Verbindung zum Netzwerk, der Mauszeiger liegt auf der Box.
- Schwarzer Rahmen: Das Netzwerk ist nicht ausgewählt.
- Blauer Rahmen: Das Netzwerk ist ausgewählt.

Der Benutzer hat nur die Möglichkeit, kein oder ein Netzwerk auszuwählen.

Der WIFI-Container hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout vom Aufbau identisch. Dafür ist das 4:3-Layout um einiges kompakter und es ändern sich nur ein paar Parameter wie z.B. die Schriftgröße.

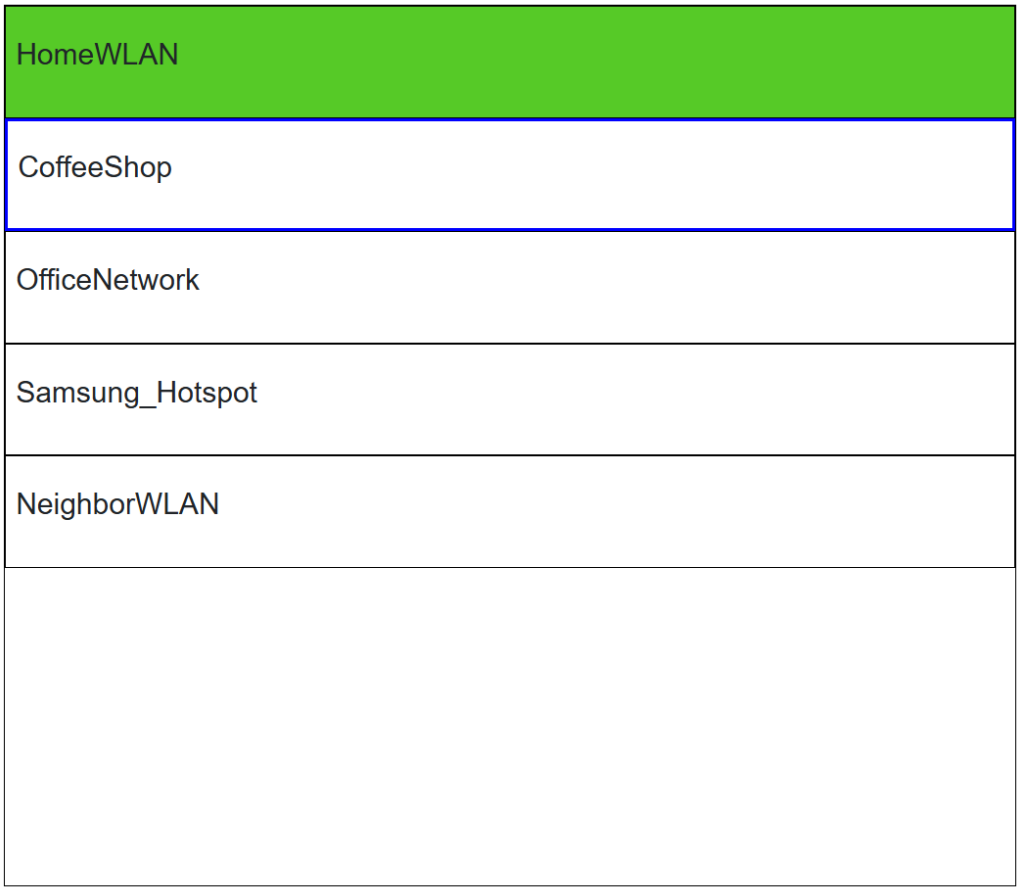

Abbildung 41: 16:9-Bildschirm WIFI-Container

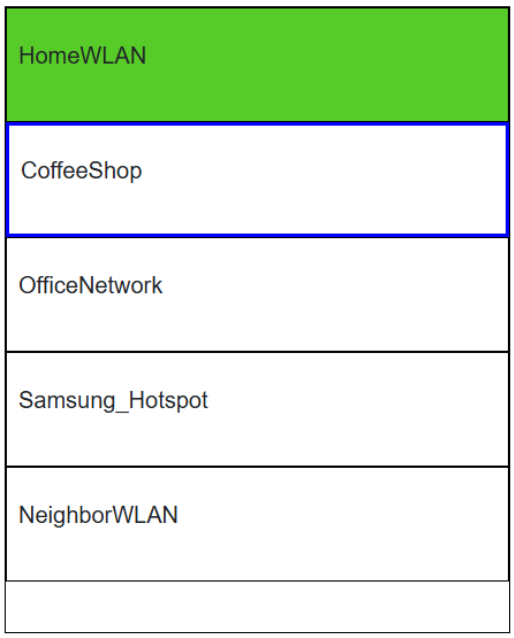

Abbildung 42: 4:3-Bildschirm WIFI-Container

Umsetzung: Zuerst werden Imports und Injects gemacht, diese erfolgen wie im Homescreen. Um die WLANs anzuzeigen, wird im HTML (siehe Listing 61) das oben Beschriebene erstellt.

Listing 61: WIFI-Container HTML

```
<div class="wifi-list">
    @if (wifiNetworks == null)
    {
        <h2>Loading...</h2>
    }
    else
    {
        foreach (var wifi in wifiNetworks.OrderByDescending(w => w.IsConnected))
        {
            <div class=@(GetWifiItemCssClass(wifi)) @onclick="() => SelectWifi(wifi)">
                <p>@wifi.Ssid</p>
            </div>
        }
    }
</div>
```

Dabei wird abgefragt, ob das Array der Netzwerke noch leer ist. Falls dies der Fall ist, wird der Text „Loading...“angezeigt. Ansonsten wird für jedes WLAN ein HTML-Element erstellt. In diesem Element steht die SSID des Netzwerks. Die Klasse des Elements wird über die Methode GetWifiItemCssClass() bestimmt. In dieser wird abgefragt, ob der RaspiSniffer mit dem WLAN verbunden ist. Danach wird abgefragt, ob das WLAN ausgewählt ist. Somit existieren vier mögliche Klassen, diese sind „connected-wifi-item selected-wifi-item“, „connected-wifi-item“, „wifi-item selected-wifi-item“ und „wifi-item“.

Bei SelectWifi() wird bei einem WLAN abgefragt, ob die SSID die gleiche ist wie die des ausgewählten WLANs. Falls dies der Fall ist, wird der String „selectedWifi“ auf „null“ gesetzt. Falls dies nicht der Fall ist, wird der String mit der SSID des WLANs überschrieben.

In der Methode OnInitializedAsync() wird die Methode PeriodicUpdate() aufgerufen. Diese Methode ruft alle 15 Sekunden die Methode LoadWifiNetworks() (siehe Listing 62) auf. Diese Methode holt die möglichen WLAN-Verbindungen von der API. Diese werden in einem Array zwischengespeichert. Danach folgt eine Abfrage, die überprüft, ob in dem Array mindestens ein WLAN gespeichert ist. Falls dies der Fall ist, folgt eine weitere Abfrage, die überprüft, ob das erste Element des Arrays nicht „null“ ist. Falls dies der Fall ist, wird das Array der Netzwerke mit dem zwischengespeicherten Array überschrieben. Ansonsten wird ein Informations-Pop-Up (8.3.2) angezeigt mit dem Text „Backend could not be reached!“.

Weiter bietet der WIFI-Container die Möglichkeit, die Netzwerke mittels der Methode GetNetworks() abzufragen und das ausgewählte Netzwerk mittels der Methode GetSelected() abzufragen.

Listing 62: LoadWifiNetworks() Methode

```
1  private async Task LoadWifiNetworks()
2  {
3      Wifi[] wifis = await ApiService.GetWifis();
4      if (wifis.Length > 0)
5      {if (wifis[0] != null)
6          {
7              wifiNetworks = wifis;
8              userWasInformed = false;
9          }
10         else if (!userWasInformed)
11         {
12             await PopUpMessage("Backend could not be reached!");
13             informationPopUp.Show();
14             userWasInformed = true;
15         }
16     }
17     StateHasChanged();
18 }
```

WIFI-Information-Container

Der WIFI-Information-Container, Klasse WIFIInformationContainer.razor, ist die Komponente, die die Informationen und Antworten der WLAN-Anfragen an die Backend-API anzeigt. Diese werden mit einem Zeitpunkt im Format „yyyy-MM-dd HH:mm:ss“ versehen. Dabei werden die aktuellsten Nachrichten unten angezeigt. Der Hintergrund des WIFI-Information-Containers ist schwarz und die Textfarbe ist weiß.

Damit der Nutzer keine Benachrichtigung übersieht, muss dieser von alleine nach unten scrollen.

Diese Nachrichten werden nur bei der Weboberfläche angezeigt, bei der der API-Aufruf getätigt wurde.

Der WIFI-Information-Container hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout vom Aufbau identisch. Dafür ist das 4:3-Layout um einiges kompakter und es ändern sich nur ein paar Parameter wie z.B. die Schriftgröße.

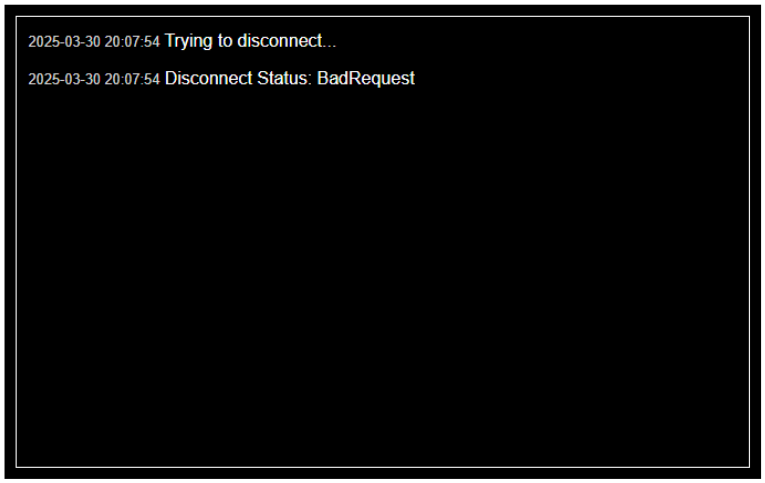

Abbildung 43: 16:9-Bildschirm WIFI-Information-Container

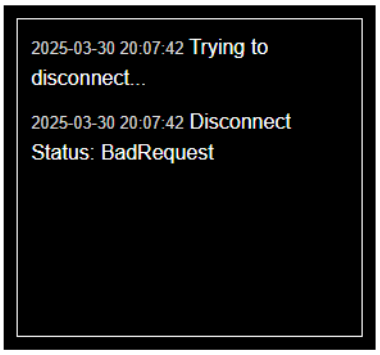

Abbildung 44: 4:3-Bildschirm WIFI-Information-Container

Umsetzung: Um die Nachrichten anzuzeigen, wird im HTML (siehe Listing 63) das oben Beschriebene erstellt.

Listing 63: WIFI-Information-Container HTML

```
<div class="wifi-information-container">
    @foreach (var log in logs)
    {
        <div class="log-entry">
            <span class="timestamp">@log.Timestamp.ToString("yyyy-MM-dd HH:mm:ss")</span>
            <span>@log.Message</span>
        </div>
    }
</div>
```

Dabei wird für jede Nachricht ein HTML-Element erstellt. Dabei wird zuerst der Zeitpunkt und dann die Nachricht angezeigt.

Der WIFI-Information-Container bietet die Möglichkeit, eine Nachricht mit der Methode Add-Log() (siehe Listing 64) hinzuzufügen. Dabei muss ein String angegeben werden, dieser ist die

Nachricht, die im WIFI-Information-Container angezeigt wird. Bei der Erstellung des Eintrags wird der gegenwärtige Moment als Zeitpunkt gespeichert.

Listing 64: AddLog() Methode

```
1 public void AddLog(string message)
2 {
3     logs.Add(new LogEntry
4     {
5         Timestamp = DateTime.Now,
6         Message = message
7     });
8     StateHasChanged();
9 }
```

Connection-Pop-Up

Das Connection-Pop-Up, Klasse ConnectionPopUp, ist ein Pop-Up, das angezeigt wird, wenn der Nutzer auf den Connect-Button auf der WIFI-Seite drückt. Es bietet die Möglichkeit, eine Verbindung zu einem WLAN herzustellen. Je nach Verschlüsselungsprotokoll muss eine SSID, ein Passwort und ein Benutzername eingegeben werden. Auch muss angegeben werden, ob es sich um ein verstecktes Netzwerk handelt. Die möglichen Verschlüsselungsprotokolle sind:

- Open: Dabei muss nur eine SSID eingegeben werden.
- WEP: Dabei muss eine SSID und ein Passwort eingegeben werden.
- WPA: Dabei muss eine SSID und ein Passwort eingegeben werden.
- WPA2: Dabei muss eine SSID und ein Passwort eingegeben werden.
- WPA3: Dabei muss eine SSID und ein Passwort eingegeben werden.
- WPA2-Enterprise: Dabei muss eine SSID, ein Passwort und ein Benutzername eingegeben werden.
- WPA3-Enterprise: Dabei muss eine SSID, ein Passwort und ein Benutzername eingegeben werden.

Standardmäßig ist „Open“ ausgewählt und die Textfelder stehen leer. Wenn aber bevor der Connect-Button gedrückt wurde ein WLAN ausgewählt wurde, wird das Verschlüsselungsprotokoll und die SSID automatisch in die entsprechenden Felder eingefügt. Der Text im Passwort-Feld wird nicht im Klartext angezeigt, sondern durch Punkte für die Buchstaben.

Solange das Pop-Up geöffnet ist, ist der Hintergrund grau.

Das Connection-Pop-Up hat nur ein Layout und ist somit im 16:9-Layout und 4:3-Layout identisch.

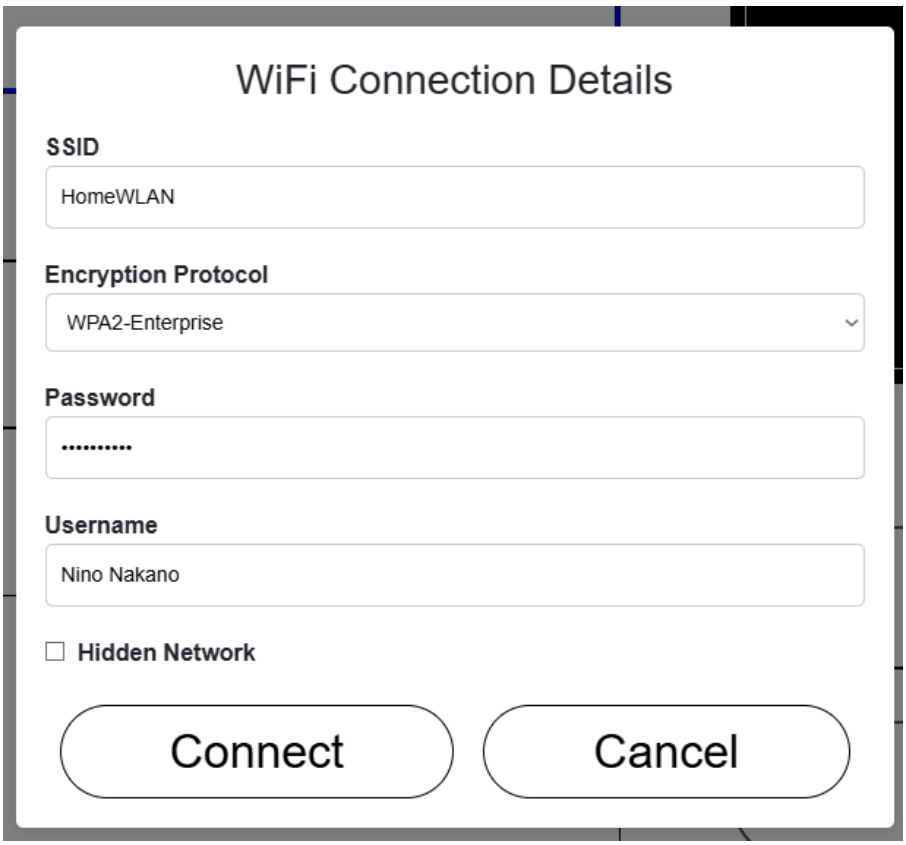

Abbildung 45: Connection-Pop-Up

Umsetzung: Zuerst werden Imports gemacht, diese erfolgen wie im Homescreen. Die Implementierung der Sichtbarkeit ist wie bei Information-Pop-Up. Um die Funktionalitäten des Pop-Ups zu verwirklichen, wird im HTML der Komponente (siehe Listing 65) das oben Beschriebene erstellt. Dabei gibt es ein Textfeld in dem die SSID eingegeben werden kann. In einem Dropdown-Menü kann das Verschlüsselungsprotokoll ausgewählt werden. Aufgrund der Auswahl ändern sich zwei Booleans. „IsPasswordRequired“ und „IsUsernameRequired“. Ersteres ist nur bei dem Verschlüsselungsprotokoll „Open“ „false“. Letzteres ist nur bei den Verschlüsselungsprotokollen „WPA2-Enterprise“ und „WPA3-Enterprise“ „true“. Dann folgen im HTML zwei Abfragen. Die erste Abfrage überprüft, ob „IsPasswordRequired“ „true“ ist. Falls dies der Fall ist, wird ein Passwortfeld für die Eingabe des Passworts angezeigt. Die zweite Abfrage überprüft, ob „IsUsernameRequired“„true“ ist. Falls dies der Fall ist, wird ein Textfeld für die Eingabe des Benutzernamen angezeigt. Darunter gibt es noch eine Checkbox, die angibt, ob das Netzwerk mit dem man sich verbinden will, versteckt ist oder nicht. Am Ende des HTMLs werden noch zwei Buttons definiert. Dabei gibt es einen Cancel-Button, die verbundene Methode Cancel() funktioniert genauso wie die Methode Close() im Information-Pop-Up. Es gibt den Connect-Button, dieser ist deaktiviert, solange die SSID leer ist. Wenn der Connect-Button gedrückt wird, wird die Methode Connect() ausgeführt. Diese führt den EventCallback OnConnect aus,

der die in der Erstellung der Komponente angegebene Methode ausführt. Dabei werden die Verbindungsdetails mitgeschickt. Danach wird das Pop-Up geschlossen.

Listing 65: Connection-Pop-Up HTML

```
@if (isVisible)
{
    <div class="popup-overlay">
        <div class="popup">
            <h3>WiFi Connection Details</h3>
            <div class="popup-content">
                <label>SSID</label>
                <input type="text" @bind="SSID" />

                <label>Encryption Protocol</label>
                <select @bind="Encryption">
                    <option value="Open">Open</option>
                    <option value="WEP">WEP</option>
                    <option value="WPA">WPA</option>
                    <option value="WPA2">WPA2</option>
                    <option value="WPA3">WPA3</option>
                    <option value="WPA2-Enterprise">WPA2-Enterprise</option>
                    <option value="WPA3-Enterprise">WPA3-Enterprise</option>
                </select>

                @*
                //Checks if isPasswordRequired is true.
                //true: Displays the input field.
                //false: Nothing happens.
                *@
                @if (IsPasswordRequired)
                {
                    <label>Password</label>
                    <input type="password" @bind="Password" />
                }

                @*
                //Checks if isUsernameRequired is true.
                //true: Displays the input field.
                //false: Nothing happens.
                *@
                @if (IsUsernameRequired)
                {
                    <label>Username</label>
                    <input type="text" @bind="Username" />
                }

                <label>
                    <input type="checkbox" @bind="IsHidden" />
                    Hidden Network
                </label>

                <div class="button-container">
                    <button class="button" @onclick="Connect"
                        disabled="@SSID.Equals("")">Connect</button>
                    <button class="button cancel" @onclick="Cancel">Cancel</button>
                </div>
            </div>
        </div>
    </div>
}
```

Damit das Pop-Up angezeigt werden kann, beinhaltet es die Methode Show(). Diese Methode setzt Standardwerte für die SSID, das Verschlüsselungsprotokoll, den Benutzernamen und das Passwort. Dieser Methode kann eine SSID und ein Verschlüsselungsprotokoll übergeben werden. Diese werden dann in die jeweiligen Variablen eingesetzt.

8.3.6 Services

ApiService

Das API-Service, Klasse ApiService.cs, ist für die Kommunikation mit dem Backend und der Geschäftslogik zuständig. Dafür wird die Klasse HttpClient verwendet, um die API-Aufrufe an das Backend zu senden. Das API-Service umfasst:

- GetIP: Die Webadresse wird aus der URL entnommen und angepasst. Diese Methode gibt die Adresse als String an den Aufrufer zurück.
- GetPackets: Es wird ein GET-Request an das Backend, Adresse „/api/packets“, gesendet. Dabei wird ein Array aus den auf dem RaspiSniffer zwischengespeicherten Paketen abgefragt und als Array an den Aufrufer zurückgegeben.
- GetRecordings: Es wird ein GET-Request an das Backend, Adresse „/api/recordings“, gesendet. Dabei wird ein Array aus Strings der Namen der Recording-Dateien, die auf dem RaspiSniffer gespeichert sind, abgefragt und als Array an den Aufrufer zurückgegeben.
- StartRecording: Es wird ein POST-Request an das Backend, Adresse „/api/startRec“, mit einem Body gesendet. Der Body ist ein Array aus Strings, der folgende vier Parameter enthält:
  - Filter-String: Ein String in der die auf die Aufzeichnung zu verwendenden Filter enthält.
  - Netzwerk: Ein String in dem steht welcher Netzwerk abgehört werden soll.
  - Anfangszeitpunkt: Ein String eines DateTime, dass den Startzeitpunkt einer Aufzeichnung angibt. Dieser Parameter ist optional.
  - Endzeitpunkt: Ein String eines DateTime, dass den Endzeitpunkt einer Aufzeichnung angibt. Dieser Parameter ist optional.

  Dabei wird eine Aufzeichnung gestartet oder geplant.
- StopRecording: Es wird ein GET-Request an das Backend, Adresse „/api/stopRec“, gesendet. Dabei wird eine Aufzeichnung beendet.
- GetUsb: Es wird ein GET-Request an das Backend, Adresse „/api/usb“, gesendet. Dabei wird ein Array aus Strings der Namen der USB-Sticks, die am RaspiSniffer angesteckt sind, abgefragt und als Array an den Aufrufer zurückgegeben.

- DismountUsb: Es wird ein POST-Request an das Backend, Adresse „/api/usb“, mit einem Body gesendet. Der Body ist ein Array aus Strings, der die Namen der ausgewählten USB-Sticks enthält. Dabei werden diese vom RaspiSniffer ausgeworfen.
- ToUsb: Es wird ein POST-Request an das Backend, Adresse „/api/usbcpy“, mit einem Body gesendet. Der Body ist ein Array aus Strings, der folgende Parameter enthält:
  - USB-Stick: Der Name des USB-Sticks auf den die Dateien kopiert werden sollen.
  - Pfad: Der Pfad, auf den die Dateien kopiert werden sollen.
  - Dateien: Alle weiteren Strings beinhalten den Namen einer Datei die kopiert werden soll.

  Dabei werden Dateien auf einen USB-Stick kopiert.
- CancelUsb: Es wird ein GET-Request an das Backend, Adresse „/api/stopcpy“, gesendet. Dabei wird der Vorgang des Kopierens auf einen USB-Stick abgebrochen.
- DeleteRecording: Es wird ein DELETE-Request an das Backend, Adresse „/api/recordings“, mit einem Body gesendet. Der Body ist ein Array aus Strings, der die Namen der ausgewählten Dateien enthält. Dabei werden die ausgewählten Dateien vom RaspiSniffer gelöscht.
- GetWifis: Es wird ein GET-Request an das Backend, Adresse „/api/wifi“, gesendet. Dabei wird ein Array aus den für den RaspiSniffer verfügbaren Netzwerkverbindungen in vereinfachter Version abgefragt und als Array an den Aufrufer zurückgegeben.
- GetWifi: Es wird ein GET-Request an das Backend, Adresse „/api/wifi“, mit einem Body gesendet. Der Body beinhaltet einen String, der die SSID des Netzwerkes enthält, das abgefragt werden soll. Dabei wird das Netzwerk mit der SSID an den Nutzer zurückgegeben.
- DisconnectWifi: Es wird ein GET-Request an das Backend, Adresse „/api/disconnect“, gesendet. Dabei wird die Verbindung zum verbundenen WLAN beendet.
- ConnectWifi: Es wird ein PUT-Request an das Backend, Adresse „/api/connect“, mit einem Body gesendet. Der Body ist ein Array aus Strings, der folgende vier Parameter enthält:
  - Verstecktes WLAN: Ein String der entweder „True“ oder „False“ enthält. Gibt an ob eine WLAN versteckt ist.
  - SSID: Ein String der die SSID des WLANs beinhaltet.

  - Passwort: Ein String der das Passwort zu einem WLAN beinhaltet.
  - Benutzername: Ein String der den Benutzernamen zu einem WLAN beinhaltet.

  Dabei wird versucht, eine Verbindung zu einem WLAN aufzubauen.

- GetTotalStorage: Es wird ein GET-Request an das Backend, Adresse „/api/totalStorage“, gesendet. Dabei wird ein Double des ganzen Speichers des RaspiSniffer abgefragt und an den Aufrufer zurückgegeben.
- GetUsedStorage: Es wird ein GET-Request an das Backend, Adresse „/api/usedStorage“, gesendet. Dabei wird ein Double des belegten Speichers des RaspiSniffer abgefragt und an den Aufrufer zurückgegeben.
- GetConnections: Es wird ein GET-Request an das Backend, Adresse „/api/networks“, gesendet. Dabei wird ein Array an Strings von den Verbindungen des RaspiSniffer abgefragt und an den Aufrufer zurückgegeben.
- IsRecording: Es wird ein GET-Request an das Backend, Adresse „/api/isrecording“, gesendet. Dabei wird der Cancellation-Token abgefragt, dieser ist False, falls gerade eine Aufzeichnung läuft oder geplant ist und umgekehrt, somit wird dem Aufrufer der gegenteilige Wert zurückgegeben.
- Schedule: Es wird ein GET-Request an das Backend, Adresse „/api/schedule“, gesendet. Dabei wird ein Array an Strings bestehend aus Startzeitpunkt und Endzeitpunkt einer geplanten Aufzeichnung an den Aufrufer zurückgegeben.

Sniff-Log-Service

Das Sniff-Log-Service, Klasse SniffLogService.cs, ist eine Klasse, die Werte für eine Session speichert. Das macht sie, indem sie in Program.cs als Scoped registriert wird und somit nur eine Instanz pro Session existiert.

Listing 66: SniffLogService als Scoped registrieren

```
1 builder.Services.AddScoped<ISniffLogService, SniffLogService>();
```

Die Werte, die im Sniff-Log-Service gespeichert werden:

- LastClearedTimestamp: Speichert den Zeitpunkt, ab welchem Pakete im Sniff-Log angezeigt werden. Dies soll über die Session gespeichert werden, weil der Wert gleich bleiben soll, auch wenn die Seite geändert wird.
- isRunning: Speichert, ob gerade eine Aufzeichnung läuft.

- sniffIsPlanned: Speichert, ob eine Aufzeichnung geplant ist.
- startTime: Speichert den Startzeitpunkt für eine geplante Aufzeichnung.
- endTime: Speichert den Endzeitpunkt für eine geplante Aufzeichnung.

8.3.7 Herausforderungen

Zwei Layouts

Für die Weboberfläche sollen zwei Layouts verwendet werden. Eines für einen 16:9-Bildschirm und eines für einen 4:3-Bildschirm. Für die Umsetzung dieses Vorhabens wurden mehrere Methoden ausprobiert.

- 1. Ansatz: Der erste Ansatz war, zwei Seiten zu entwickeln, die die gleichen Funktionen erfüllen, und zwischen diesen zu wechseln falls die Bedingungen zum Wechseln gegeben sind.
  Problem: Diese Lösung wäre sehr umständlich und hätte viele Probleme verursacht. Dazu wäre es ein großer Mehraufwand, da alles doppelt programmiert werden hätte müssen.
- 2. Ansatz: Der zweite Ansatz war, anstatt zwei Seiten zu entwickeln, zwei Komponenten zu entwickeln, die beide im Code der Seite implementiert werden. Die Idee war, zwischen den beiden Komponenten zu wechseln, falls die Bedingungen zum Wechseln gegeben wären.
  Problem: Ähnlich wie beim ersten Ansatz.
- 3.Ansatz: Der dritte Ansatz war, ansattt zwei Seiten oder zwei Komponenten zu entwickeln, zwei CSS zu entwickeln. In dem Fall würde im Code nur das CSS ausgetauscht. Somit würden nur zwei Design pro Seite gebraucht werden.
  Problem: Es müssen immernoch zwei CSS Files erstellt werden wobei vieles doppelt vorkommen würde.
- 4. Ansatz: Der vierte und finale Ansatz war, @media() zu verwenden. @media() ist eine CSS-Regel, die es ermöglicht, Stilregeln basierend auf verschiedenen Medienabfragen oder Bildschirmbedingungen anzuwenden. Es ermöglicht das Design und Layout einer Webseite je nach den Eigenschaften des Geräts oder des Browsers anzupassen.[29] [29]

[29] *https://www.w3schools.com/cssref/atrule_media.php*

Listing 67: @media Beispiel

```
.sniff-log {
    height: 100%;
    overflow-y: auto;
    border: 1px solid black;
}

.packet {
    border: 1px solid black;
    height: 8%;
    min-height: 60px;
    font-size: 20px;
    display: flex;
    align-items: center;
}

.packet-text{
    white-space: normal;
}


@media (max-width: 800px) {
    @media (max-height: 600px) {
        .packet {
            min-height: 20px;
            font-size: 15px;
            align-items: baseline;
        }
    }
    .packet-text{
        white-space: nowrap;
    }
}
```

Dieses Beispiel zeigt, dass bei bestimmten Bedingungen bestimmte HTML-Elemente der Klasse „packet“ abgeändert werden. Diese Lösung ist einfach zu implementieren, erzeugt keine weiteren Files oder doppelten Code, der nicht benötigt wird.
Diese Lösung wird für die Implementierung der verschiedenen Layouts verwendet.

Zeitpunktspeicherung

Es soll ein Zeitpunkt gespeichert werden, ab welchem Pakete angezeigt werden sollen. Dies soll passieren, damit z.B. auf verschiedenen PCs und dem RaspiSniffer die Sniff-Logs verschieden lang sein können. Weiters sollen Pakete im Sniff-Log angezeigt werden, die aufgenommen wurden, während der Nutzer sich nicht im Homescreen aufhielt.
Ein Zeitpunkt ist gespeichert und nur Pakete, die nach diesem Zeitpunkt aufgenommen wurden, sollen im Sniff-Log angezeigt werden.
Wenn der Clear-Button gedrückt wird, wird der Zeitpunkt auf den Moment des Clears geändert. Der Zeitpunkt soll über die Pages hinweg beibehalten werden.

- 1. Ansatz Der erste Ansatz war, den Zeitpunkt einfach im Code zu speichern und in OnInitialized() immer wieder zu aktualisieren.
  Problem: Jedes Mal wenn die Seite initialisiert wird, geht der Zeitpunkt verloren und wird neu gesetzt.

- 2. Ansatz Der zweite Ansatz war die Erstellung des SniffLogService.cs, das Registrieren als Scoped im Program.cs und das Injecten des Services in der Home.razor Klasse. Da dieser beim Start der App als Scoped registriert wird, wird der Zeitpunkt einmal für die Session, bei der Initialisierung der Klasse Home.razor angelegt und wird nur geändert falls der Clear-Button auf der Hauptseite geklickt wird. [30] [30]

Diese Lösung wird für die Implementierung der Zeitpunktspeicherung verwendet.

Adresse im Frontend

Nach der Erstellung des ApiService.cs kam die Frage auf, wie man die Adresse für die API dynamisch im ApiService.cs eingetragen werden kann. Anfangs war die Idee, dass das Backend die Adresse in einer Datei auf dem Raspberry Pi ablegt und das Frontend die Datei ausliest und die Adresse in einem String speichert.

- 1. Ansatz: Der erste Ansatz war das Auslesen der Datei über die StreamReader Klasse. Problem: Der Streamreader hat egal was man versucht hat die Datei nicht gefunden.
- 2. Ansatz: Der zweite Ansatz war das Auslesen der Datei über die File Klasse. Problem: Sowie beim StreamReader wurde die Datei nicht gefunden, somit konnte die Datei auch nicht ausgelesen werden.
- 3. Ansatz: Der dritte Ansatz war das Auslesen der Datei über die HttpClient Klasse. Bei dieser Methode wurde die Datei gefunden und konnte ausgelesen werden. Somit konnten API-Aufrufe durchgeführt werden.
  Problem: Der Prozess läuft asynchron somit wird die Adresse erst dann geholt, wenn die Seite schon angezeigt und geladen wurde. Das hatte zur Folge das die Adresse nicht angezeigt wird.
- 4. Ansatz: Der vierte Ansatz war es die BaseAddress von HttpClient zu nutzen. Dabei wird die Basisadresse der Internetressource abgerufen. [31] [31] Dies ist in diesem Fall möglich, da die Adresse für die Weboberfläche und die API über die Gleiche ist. Diese Lösung wird für die Implementierung der verschiedenen Layouts verwendet.

Diese Lösung wird für die Implementierung der Findung der Adresse im Frontend verwendet.

[30] *https://learn.microsoft.com/en-us/aspnet/core/blazor/fundamentals/dependency-injection?view=aspnetcore-9.0*
[31] *https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.baseaddress?view=net-8.0*

9 Ergebnis

Mit dem RaspiSniffer können Verbindungen und Netzwerke aufgezeichnet werden. Diese Aufzeichnungen werden auf dem RaspiSniffer gespeichert. Diese können dann über die Weboberfläche auf einen USB-Stick oder auf ein verbundenes Gerät kopiert werden. Die WLAN-Verbindung kann über die Weboberfläche angepasst werden. Informationen zum Verbindungsaufbau oder Verbindungstrennung können auch eingesehen werden. Weiter kann das Ergebnis dieser Diplomarbeit in zwei Teile aufgeteilt werden. Die API und das Frontend.

9.1 API

Die API, welche das Backend dieser Diplomarbeit darstellt, ist direkt in das Frontend integriert. Über die von der API angebotenen Endpoints ist es Nutzern, die sich im selben Netzwerk wie der RaspiSniffer befinden, möglich, das Gerät zu steuern und Daten abzufragen.

In der finalen Version übertrifft die API die ursprünglichen Anforderungen. Sie implementiert nicht nur alle benötigten Endpoints, sondern stellt darüber hinaus zusätzliche Funktionen zur Verfügung. So war beispielsweise die Möglichkeit, Sniffing-Vorgänge zu planen, nicht Teil der ursprünglichen Spezifikation.

9.2 Frontend

Das Frontend bietet eine Weboberfläche, die zwei verschiedene Layouts hat, zwischen denen einfach gewechselt werden kann. Dabei gibt es ein 16:9-Layout und ein 4:3-Layout. Das Design ist simpel gehalten, damit die Nutzung der Weboberfläche auf einem kleinen Raspberry Pi Bildschirm nicht eingeschränkt wird.

In der finalen Version übertrifft die Weboberfläche die ursprünglichen Anforderungen. Sie implementiert nicht nur alle geforderten Funktionalitäten, sondern bietet darüber hinaus zusätzliche Features. So war beispielsweise die Möglichkeit, Dateien auf ein verbundenes Gerät herunterzuladen, nicht Teil der ursprünglichen Spezifikation.

10 Resümee

Dank der Erfahrung aus dem Projektunterricht und der Unterstützung des Auftraggebers konnte die Planungsphase inklusive Technologien und Ausweichmöglichkeiten innerhalb der ersten Praktikumswoche absolviert werden. Als Vorgehensmodell haben wir uns für das hybride Vorgehensmodell Scrumban entschieden. Dieses konnte perfekt in einem vierwöchigen Praktikum im Sommer 2024 bei der Kontron AG in Linz angewendet werden.

Im Rahmen der Durchführung der Diplomarbeit hatten wir die Möglichkeit, neue Technologien kennenzulernen und anzuwenden. Dies war durch eigenes Erarbeiten und die Unterstützung des Auftraggebers möglich. Die folgenden Themen zählen zu diesen:

- Netsniff-ng
- Nginx
- Rider
- CLI.Wrap
- ASP.NET Core
- Blazor WebAssembly
- Bash

Der Anfang brachte Schwierigkeiten, da der originale Auftraggeber der Arbeit, Secureguard von der Kontron AG, aufgekauft wurde. Da die Person, die die Idee für die Diplomarbeit einbrachte, das Projekt in der Planungsphase verlassen hatte, mussten wir somit mit dem neuen Auftraggeber wieder auf den gleichen Nenner kommen. Diese Hürde wurde aber schnell überwunden, indem mehrere Meetings vor und im Praktikum gehalten wurden. Somit konnte nach der Planungsphase, noch im Praktikum, das Projekt ohne große Komplikationen mit allen MVP-Features und Nice-To-Have-Features abgeschlossen werden.

Glossar

API Application Programming Interface

ASP.NET Active Server Pages Network Enabled Technologies

Bash Bourne Again Shell

CLI Command Line Interface

CMD Command Prompt

CORS Cross-Origin Resource Sharing

CPU Central processing unit

CSS Cascading Style Sheets

dccp Datagram Congestion Control Protocol

GB Gigabyte

GNU GNU's Not Unix

HDMI High-Definition Multimedia Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IoT Internet of Things

IP Internet Protocol

IPv4 Internet Protocol version 4

JSON JavaScript Object Notation

KiB Kibibyte

LAN Local Area Network

MAC medium access control address

MITM Man-in-the-Middle

MVC Model-View-Controller

MVP Minimum Viable Product

OS Operating System

PC Personal Computer

pcap Packet Capture

RAM Random access memory

sctp Stream Control Transmission Protocol

SD Secure Digital

SSH Secure Shell Protocol

SSID Service Set Identifier

TCP Transmission Control Protocol

UDP User Datagram Protocol

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

USB Universal Serial Bus

VPN Virtual Private Network

WEP Wired Equivalent Privacy

WLAN Wireless Local Area Network

WPA Wi-Fi Protected Access

Literaturverzeichnis

[1] Kontron, „Über uns Kontron.” Online verfügbar: https://www.kontron.com/de/konzern/ueber-uns

[2] Cloudflare, „What is a packet? | Network packet definition,” letzter Zugriff am 20.03.2025. Online verfügbar: https://www.cloudflare.com/learning/network-layer/what-is-a-packet/

[3] Imperva, „Man in the middle (MITM) attack,” letzter Zugriff am 17.03.2025. Online verfügbar: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

[4] Fortinet, „Man-in-the-Middle-Angriff: Arten und Beispiele,” letzter Zugriff am 18.03.2025. Online verfügbar: https://www.fortinet.com/de/resources/cyberglossary/man-in-the-middle-attack

[5] Refactoring.Guru, „Singleton,” letzter Zugriff am 19.03.2025. Online verfügbar: https://refactoring.guru/design-patterns/singleton

[6] Medium, „Navigating the Complex World of Singletons in Multithreaded Applications,” letzter Zugriff am 19.03.2025. Online verfügbar: ttps://medium.com/@berktorun.dev/navigating-the-complex-world-of-singletons-in-multithreaded-applications-867bb03eed02

[7] Microsoft, „C Artikel von Microsoft.” Online verfügbar: https://dotnet.microsoft.com/en-us/languages/csharp

[8] ——, „APS.NET Core Artikel von Microsoft.” Online verfügbar: https://dotnet.microsoft.com/en-us/apps/aspnet

[9] ——, „Blazor Artikel von Microsoft.” Online verfügbar: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

[10] Raspberry Pi Foundation, „Raspberry Pi.” Online verfügbar: https://www.raspberrypi.com/

[11] Wikipedia, „Raspberry Pi Modelle.” Online verfügbar: https://de.wikipedia.org/wiki/Raspberry_Pi#Raspberry-Pi-Modelle

[12] ——, „Raspberry Pi OS.” Online verfügbar: https://de.wikipedia.org/wiki/Raspberry_Pi#Raspberry_Pi_OS

[13] Free Software Foundation, „Bash Scripting Language.” Online verfügbar: https://www.gnu.org/software/bash/

[14] Tcpdump, „Tcpdump.” Online verfügbar: https://www.tcpdump.org/

[15] Unabhängige Gruppe an Programmierer, „Sniffnet.” Online verfügbar: https://www.tcpdump.org/

[16] D. Borkmann, „Netsniff-ng.” Online verfügbar: http://netsniff-ng.org/

[17] Wikipedia, „Zero-Copy.” Online verfügbar: https://en.wikipedia.org/wiki/Zero-copy

[18] „Recieve-Ringe.” Online verfügbar: https://pdos.csail.mit.edu/6.828/2021/labs/net.html

[19] Y. Gurtovoy, „Slsnif.” Online verfügbar: https://linux.die.net/man/1/slsnif

[20] I. Sysoev, „Nginx.” Online verfügbar: https://nginx.org/

[21] JetBrains, „ReSharper.” Online verfügbar: https://www.jetbrains.com/resharper/

[22] ——, „Rider.” Online verfügbar: https://www.jetbrains.com/rider/

[23] C. Allegretta, „Nano.” Online verfügbar: https://www.nano-editor.org/

[24] O. Holub, „CliWrap.” Online verfügbar: https://github.com/Tyrrrz/CliWrap

[25] F. Keller, „BerryBase Liste and möglichen Betriebssystem für Raspberry Pi.” Online verfügbar: https://blog.berrybase.de/raspberry-pi-die-wichtigsten-7-betriebssysteme-im-vergleich/

[26] Microsoft, „Hosting ASP.Net Blazor WebAssembly.” Online verfügbar: https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/webassembly?view=aspnetcore-9.0

[27] ——, „Verarbeiten von Anforderungen mit Controllern in ASP.NET Core MVC.” Online verfügbar: https://learn.microsoft.com/de-de/aspnet/core/mvc/controllers/actions?view=aspnetcore-9.0

[28] ——, „Blazor Webassembly,” Was ist Blazor Webassembly? Online verfügbar: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-9.0

[29] w3schools, „@media Regel,” Beschreibung @media Regel. Online verfügbar: https://www.w3schools.com/cssref/atrule_media.php

[30] Microsoft, „Scoped Service,” Scoped Service hinzufügen. Online verfügbar: https://learn.microsoft.com/en-us/aspnet/core/blazor/fundamentals/dependency-injection?view=aspnetcore-9.0

[31] ——, „HttpClient.BaseAddress,” Was ist HttpClient.BaseAddress? Online verfügbar: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.baseaddress?view=net-8.0

Abbildungsverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

A.1 Thomas Hintersteininger

1. Abstract (4)
2. Zusammenfassung (5)
3. Auftraggeber (6.1)
4. Motivation (6.2)
5. Ausgangslage (6.3)
6. Projektinhalt (6.4)
7. Frontend (8.3)
8. Resümee (10)

A.2 Samuel Schlöglhofer

1. Aufbau und Struktur (6.6)
2. Fachbegriffe (7.1)
3. Verwendete Bibliotheken (7.4)
4. Backend (8.2)
5. Ergebnis (9)

A.3 Daniel Kertesz

1. Projektumfeld (6.5)
2. Verwendete Technologien (7.2)
3. Verwendete Entwicklungsumgebungen (7.3)
4. Verwendete Hardware (7.5)
5. Raspberry Pi (8.1)

B Plakat

Abbildung 46: Diplomarbeitsplakat