HÖHERE TECHNISCHE BUNDESLEHRANSTALT Wien 16
Abteilung für Informationstechnologie

DIPLOMARBEIT

LinkWave

Ausgeführt im Schuljahr 2023/24

Arshia Reisi (5CHITN) Betreuer: ***

Jan Schäfer (5AHITN) Betreuer: ***

Ruben Zukić (5AHITN) Betreuer: ***

Selbstständigkeitserklärung

Wir erklären, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

Wien, am TT.MM.JJJJ

Verfasser / Verfasserinnen:

Ruben Zukić

Jan Schäfer

Arshia Reisi

Dokumentation der Diplomarbeit

Verfasser	Arshia Reisi Jan Schäfer Ruben Zukić
Jahrgang / Klasse Schuljahr	5XHITN / 2023-24
Thema	Erstellung einer plattformübergreifenden Anwendung zur Optimierung der Daten- und Informationsübertragung zwischen Geräten unterschiedlicher Betriebssysteme

Abgabevermerk	Datum: ***	Übernommen von:
Approbation	Datum: ***	Prüfer:
	Datum: ***	Abteilungsvorstand: ***

Kurzfassung

Kennen Sie AirDrop? AirDrop ist eine Funktion der Apple-Betriebssysteme, die es ermöglicht, Daten schnell und einfach zwischen Apple-Geräten zu teilen. Bilder und Dateien können mittels Knopfdruck auf andere Geräte übertragen werden und selbst die Zwischenablage der eigenen Geräte wird im Hintergrund synchronisiert.

LinkWave greift diese Funktionsweise der Apple-Geräte auf und erweitert sie auf eine betriebssystemunabhängige Ebene. LinkWave ist eine innovative Softwarelösung, die es ermöglicht, AirDrop-ähnliche „Connectivity-Features“ auf einer breiten Palette von Plattformen zu nutzen, einschließlich Windows, Android und den Apple-Geräten selbst. Die Anwendung ermöglicht den nahtlosen Austausch von Dateien, Bildern und Zwischenablageinhalten über unterschiedliche Betriebssysteme hinweg, ohne dass die Benutzerinnen und Benutzer auf die Kompatibilität der Geräte achten müssen.

Abstract

Have you heard about AirDrop? AirDrop is a function of Apple operating systems that makes it possible to share data quickly and easily between Apple devices. Pictures and files can be sent to other devices at the touch of a button and even the clipboard of your own devices is synchronized in the background.

LinkWave takes this functionality from Apple devices and extends it to an operating system-independent level. LinkWave is an innovative software solution that enables AirDrop-like connectivity across a wide range of platforms, including Windows, Android and Apple devices themselves. The application enables the seamless sharing of files, images and clipboard content across different operating systems without users having to worry about device compatibility.

Inhaltsverzeichnis

1 Einleitung

Der Arbeitsalltag unserer modernen Welt wird immer schneller, mobiler und digitaler. Dank dieser Entwicklung ist es heute möglich, von überall auf der Welt, zu jeder Zeit nach Belieben zu arbeiten. Um diese Flexibilität zu erleben, sind für den Arbeitsalltag Geräte wie Computer, Smartphones und Tablets unverzichtbare Werkzeuge geworden. Sie ermöglichen es, wichtige Daten und Informationen jederzeit und überall abzurufen, zu bearbeiten und zu teilen.

1.1 Problemstellung

Viele Menschen benutzen mittlerweile mehrere Geräte, um ihre Arbeit zu erledigen - Dabei kommen oft Geräte von vielen verschiedenen Herstellern zum Einsatz. Mit dem zunehmenden Einsatz verschiedener Geräte wird es jedoch immer komplizierter, diese effizient miteinander zu benutzen.

Vor allem das Teilen von Daten und Informationen zwischen Geräten ist oft umständlich und zeitaufwendig. Um darzustellen, welche konkreten Probleme häufig auftreten, werden im Folgenden einige Beispiele genannt:

- **Umständliches Übertragen von Dateien:** Das Übertragen von Dateien zwischen verschiedenen Geräten ist vor allem bei Geräten von verschiedenen Herstellern umständlich und zeitaufwendig. Wenn man beispielsweise eine Datei von einem iPhone auf einen Windows-Computer übertragen möchte, muss man auf Cloud-Dienste wie Dropbox oder Google Drive zurückgreifen.

  Hierzu muss man aber sehr viele Arbeitsschritte durchführen. Man muss sich bei einem Cloud-Dienst anmelden, die Datei über einen Webbrowser hochladen, sich auf dem anderen Gerät wieder bei dem Cloud-Dienst anmelden und die Datei wieder herunterladen. Dafür muss man auch noch Programme herunterladen, die nichts mit dem eigentlichen Übertragen von Dateien zu tun haben. Dies führt dazu, dass das Übertragen von Dateien oft so lästig ist, dass man es lieber ganz sein lässt und sich nur auf ein Gerät beschränkt.

- **Übertragung von großen Dateien:** Vor allem in der Kreativbranche kommen oft mehrere GB große Dateien wie Bilder, Videos und Musik vor, die häufig zwischen verschiedenen Menschen und Geräten ausgetauscht werden müssen. Für kleinere Teams und Einzelpersonen bedeutet dies oft, dass sie auf Cloud-Dienste wie Dropbox oder Google Drive zurückgreifen müssen, um diese Dateien zu teilen.

  Hierbei ist man jedoch durch die Geschwindigkeit der Internetverbindung und die Größe des Speichers limitiert. Zudem müssen die Dateien erst hochgeladen und dann wieder heruntergeladen werden, was für die ohnehin schon belastete Internetverbindung doppelten Aufwand bedeutet. Man müsste zudem adäquaten Speicherplatz bei einem Cloud-Dienst erwerben, was vor allem für kleinere Teams und Einzelpersonen ein wesentlicher Kostenfaktor ist.

- **Häufiges Übertragen von Dateien:** Auch wenn es sich um kleinere Dateien handelt, die übertragen werden müssen, wird der Prozess des Hochladens und Herunterladens von Dateien über Cloud-Dienste bei häufiger Wiederholung schnell lästig.

  Diese repetitive Aufgabe kann schnell so zeitaufwendig werden, dass man seine Arbeit auf nur ein Gerät beschränkt. Dies führt dazu, dass man oft weniger flexibel arbeiten kann und anderswo Umwege suchen muss.

- **Übertragen von Texten und Bildern:** Wenn man mehrere Geräte zum Arbeiten benutzt kann es oft vorkommen, dass man Texte und Bilder zwischen diesen Geräten benutzen muss. Wenn man dies machen möchte, muss man auch auf Umwege zurückgreifen. Man könnte sich Texte und Bilder in E-Mails, oder über Nachrichtendienste wie WhatsApp oder Signal an sich selbst schicken.

  Doch vor allem für Texte, die man sehr schnell auf anderen Geräten benutzen möchte, wie beispielsweise Weblinks, ist dies oft zu umständlich. Auch eine kurze Verzögerung beim Übertragen von solchen, sehr oft benutzten, Daten kann auf Dauer schnell extrem lästig werden.

Manche Hersteller wie Apple bieten innerhalb ihres Ökosystems[1] Lösungen wie “Airdrop” und “Universal Clipboard” an, die das nahtlose Teilen von Daten innerhalb des Sortiments des Herstellers ermöglichen. Es fehlt jedoch eine ähnlich integrierte Lösung, die unabhängig von Hersteller und Betriebssystem funktioniert. Hier setzt LinkWave an, um diese Lücke zu schließen und eine universelle, benutzerfreundliche Lösung zu bieten.

1.2 Lösung

LinkWave zielt darauf ab, eine Ökosystem-Erfahrung zum herunterladen zwischen Mac-, iPhone-, Android- und Windows-Geräten herzustellen. LinkWave kann in Form einer App auf diesen Geräten installiert werden. Diese App findet automatisch andere Geräte in der Nähe, die ebenfalls LinkWave installiert haben. Zwischen diesen Geräten können sämtliche LinkWave Funktionen genutzt werden.

[1] Ökosystem - Im Kontext von Technologie bezeichnet ein Ökosystem die Gesamtheit von Geräten, Software und Diensten, die von einem Hersteller angeboten werden.

2 Funktionalität

2.1 Funktionen

LinkWave bietet eine Vielzahl von Funktionen, die es ermöglichen, Daten und Informationen zwischen verschiedenen Geräten auszutauschen. Um einen groben Überblick über die Funktionalität zu geben, werden die LinkWave Funktionen im Folgenden kurz vorgestellt:

- **Geräteerkennung:** LinkWave erkennt automatisch andere Geräte in der Nähe, die ebenfalls LinkWave installiert haben. So können beispielsweise Dateien zwischen einem Smartphone und einem Computer übertragen werden, ohne dass manuell eine Verbindung hergestellt werden muss.

- **Dateiübertragung:** LinkWave ermöglicht es, Dateien zwischen Geräten zu übertragen. Dabei spielt es keine Rolle, ob die Geräte von verschiedenen Herstellern stammen oder unterschiedliche Betriebssysteme verwenden. Die Dateiübertragung erfolgt über eine direkte Verbindung zwischen den Geräten, ohne dass eine Internetverbindung benötigt wird.

- **Zwischenablage:** LinkWave ermöglicht es, den Inhalt der Zwischenablage zwischen Geräten zu teilen. So können beispielsweise Texte oder Bilder, die auf einem Smartphone kopiert wurden, auf einem Computer eingefügt werden. Dies passiert automatisch, sobald die Geräte in der Nähe sind. Es ist aus Sicherheitsgründen jedoch nur zwischen Geräten möglich, die mit dem selben Benutzerkonto angemeldet sind.

- **Benutzerkonto:** LinkWave ermöglicht es, sich mit einem Benutzerkonto anzumelden. Dadurch werden die Einstellungen und Daten des Benutzers auf allen Geräten synchronisiert. So kann beispielsweise die Zwischenablage zwischen Geräten geteilt werden, die mit dem selben Benutzerkonto angemeldet sind.

- **Hintergrundausführung:** LinkWave kann im Hintergrund ausgeführt werden, ohne dass die App geöffnet sein muss. So können beispielsweise Dateien zwischen Geräten übertragen werden, während die App im Hintergrund läuft. Auch die Zwischenablage kann im Hintergrund geteilt werden.

- **Verschlüsselung:** LinkWave verschlüsselt alle Daten, die zwischen kommunizierenden Geräten übertragen werden. Das gewährleistet, dass niemand außer dem Benutzer die Daten einsehen kann. Die Verschlüsselung erfolgt automatisch und ist für den Benutzer unsichtbar.

2.2 Funktionalität auf verschiedenen Betriebssystemen

LinkWave unterstützt eine Vielzahl von Betriebssystemen. Aufgrund von Einschränkungen der Betriebssysteme und der Hardware, sind jedoch nicht alle Funktionen auf allen Betriebssystemen voll funktionstüchtig. In Tabelle 1 sind die unterstützten Funktionen auf verschiedenen Betriebssystemen und deren Einschränkungen aufgelistet.

	Geräteerkennung	Dateiübertragung	Zwischenablage	Benutzerkonto	Hintergrundausführung	Verschlüsselung
iOS	✓	✓	~	✓	✗	✓
macOS	✓	✓	✓	✓	✓	✓
Android	✓	✓	~	✓	✓	✓
Windows	✓	✓	✓	✓	✓	✓

✓ – Voll unterstützt, ~ – Teilweise unterstützt, ✗ – Nicht unterstützt

Tabelle 1: Unterstützte Funktionen auf verschiedenen Betriebssystemen

Die genaueren Einschränkungen und Implemetierungen auf den verschiedenen Betriebssystemen werden in Kapitel 5 “Implementierung der Funktionen” ab Seite 61 und den entsprechenden Unterkapiteln der einzelnen Funktionen beschrieben.

3 Applikationen

Die Funktionen von LinkWave werden in Form von Apps auf den verschiedenen Betriebssystemen zur Verfügung gestellt. Auf jedem unterstützten Betriebssystem wurde die App in einer nativen Programmiersprache entwickelt, um die bestmögliche Integration zu gewährleisten. Die Apps stehen auf der webseite von LinkWave `linkwave.org` zum Download bereit.

3.1 iOS

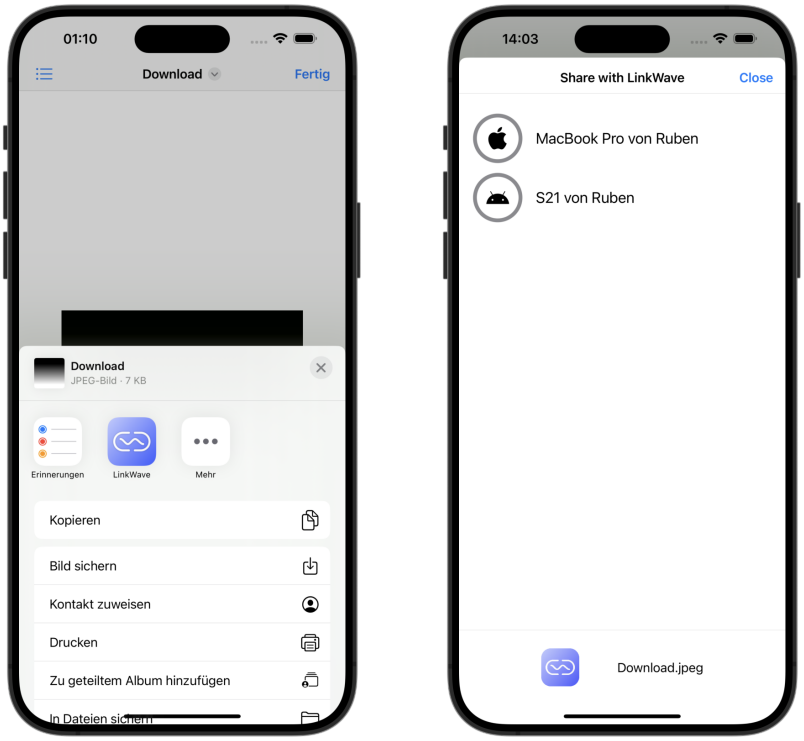
Auf iOS-Geräten wurde die LinkWave App in der Programmiersprache Swift entwickelt. Für das UI wurde das Framework SwiftUI verwendet. Swift und SwiftUI wurden von Apple entwickelt und ermöglichen es leicht eine Oberfläche, die den Apple Kriterien entspricht, zu erstellen. Die App ist für iOS 17 und neuer verfügbar.

Um Benutzer:innen die Benutzung von LinkWave so einfach wie möglich zu machen kommt die iOS App mit verschiedenen Möglichkeiten, LinkWave zu benutzen. Die implementierten Möglichkeiten sind:

- **Die LinkWave App:** In der App können Benutzer:innen die Dateiübertragung benutzen, sowie die Einstellungen der App ändern. In der App kann man sich zudem mit seinem LinkWave Konto anmelden, um mehr Funktionen frei zu schalten.

Abbildung 1: iOS LinkWave App

- **Das “Teilen” Fenster:** In iOS gibt es die Funktion “Teilen”. Diese Funktion ermöglicht es, Dateien, Texte und Bilder aus anderen Apps in andere Apps zu teilen. Man muss lediglich auf das Teilen-Symbol in einer App drücken und bekommt eine Auswahl an Apps, mit denen man den Inhalt teilen kann. LinkWave ist eine dieser Apps. So muss man die LinkWave App nicht öffnen, um Dateien zu teilen. Man drückt einfach auf das Teilen-Symbol in anderen Apps und wählt LinkWave aus. Von dort aus kann man Dateien direkt an andere Geräte senden. Texte lassen sich außerdem direkt in die Zwischenablage eines anderen Gerätes kopieren.



(a) iOS Teilen Auswahl

(b) iOS LinkWave Teilen Fenster

Abbildung 2: iOS Teilen Erweiterung

3.2 macOS

Auf macOS-Geräten wurde die LinkWave App ebenfalls in der Programmiersprache Swift entwickelt. Für das UI wird auch das Framework SwiftUI verwendet. Die Die App ist für macOS 14 und neuer verfügbar.

Die macOS App bietet genau so wie die iOS App verschiedene Möglichkeiten, LinkWave zu benutzen. Aufgrund von mehr Möglichkeiten in der Entwicklung für macOS sind diese jedoch viel weitreichender. Die implementierten Möglichkeiten sind:

- **Die LinkWave App:** In der App können Benutzer:innen genau so wie auf iOS die Dateiübertragung benutzen, sowie die Einstellungen der App ändern. In der App kann man sich auch mit seinem LinkWave Konto anmelden, um mehr Funktionen frei zu schalten. Zudem kann man eine Übersicht über alle Geräte in der Nähe sehen, die LinkWave benutzen.

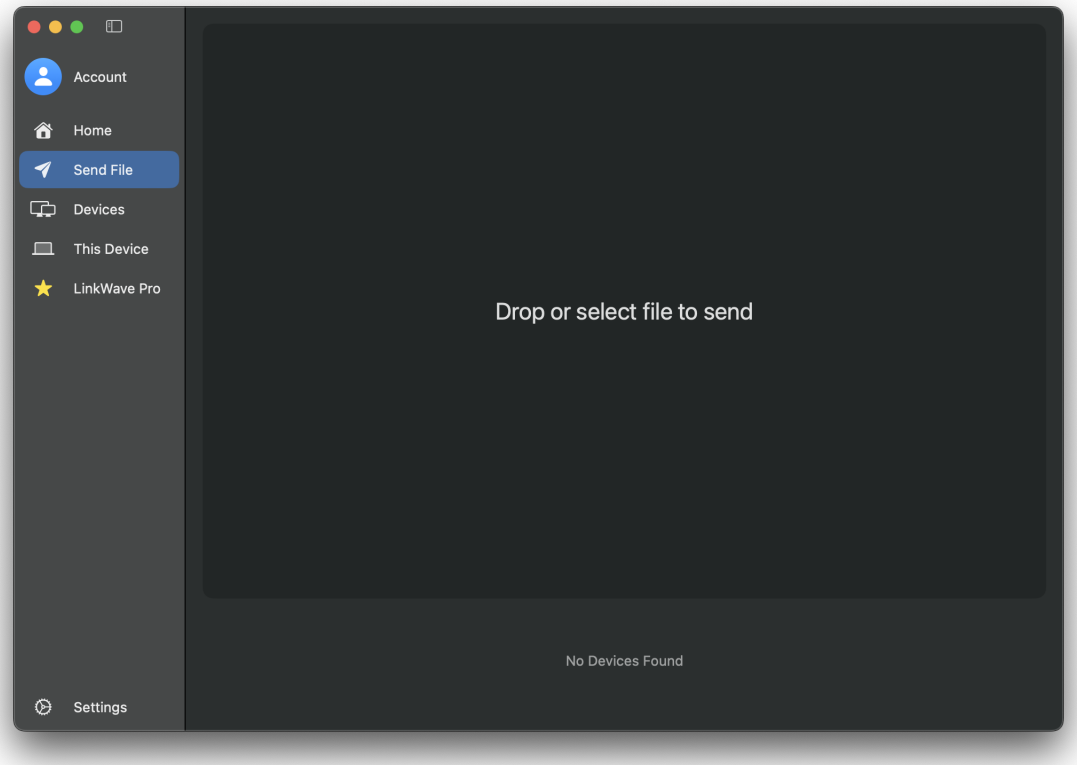
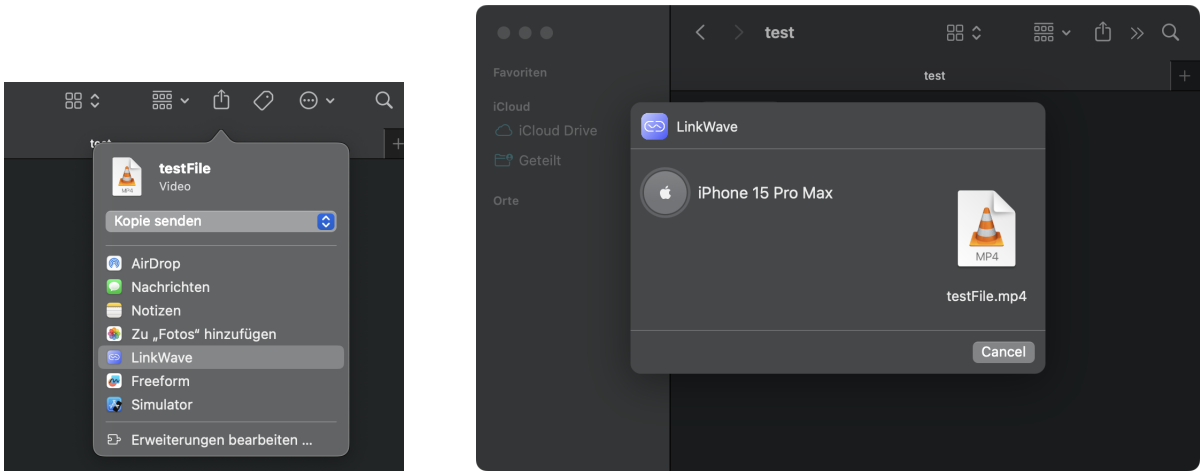


Abbildung 3: macOS LinkWave App

- **Das “Teilen” Fenster:** Auch auf macOS gibt es die Funktion “Teilen”. Diese Funktion funktioniert identisch wie auf iOS. Nur das Teilen von Texten in die Zwischenablage eines anderen Gerätes ist auf macOS nicht möglich. Dies liegt daran, dass die Zwischenablage auf macOS direkt benutzt werden kann und deswegen keine Funktion wie auf iOS benötigt wird.



(a) macOS Teilen Auswahl

(b) macOS LinkWave Teilen Fenster

Abbildung 4: macOS Teilen Erweiterung

- **Das Menüleisten-Icon:** Auf macOS gibt es die Menüleiste. In dieser Menüleiste sind die Symbole von Apps zu sehen, die im Hintergrund laufen. LinkWave hat auch ein solches Symbol. Über dieses Symbol kann man die LinkWave App und die Einstellungen öffnen. Man hat auch eine Übersicht über die mit dem eigenen Konto verbundenen Geräte und kann deren Einstellungen direkt öffnen. Man kann LinkWave auch direkt komplett beenden, so dass es nicht mehr im Hintergrund läuft.

Abbildung 5: macOS Menüleisten-Icon

- **Die Zwischenablage:** Auf macOS gibt es wie auf jedem modernen Betriebssystem eine Zwischenablage. LinkWave sendet kopierte Inhalte automatisch an andere Geräte des eigenen Kontos.

3.3 Android

Die ”LinkWave” App für Android wurde in der Programmiersprache Kotlin entwickelt. Für das UI wurde das Framework Jetpack Compose verwendet. Die App ist für Android 12 und neuer verfügbar.

Die Android App besteht aus drei Teilen:

- **Die LinkWave App:** Unter der LinkWave App ist hier die Hauptanwendung zu verstehen, also die App, die sich beim Drücken auf das App-Icon öffnet. In dieser App können Benutzer:innen Einstellungen wie die Sichtbarkeit ihres Gerätes ändern und sich an- und abmelden.

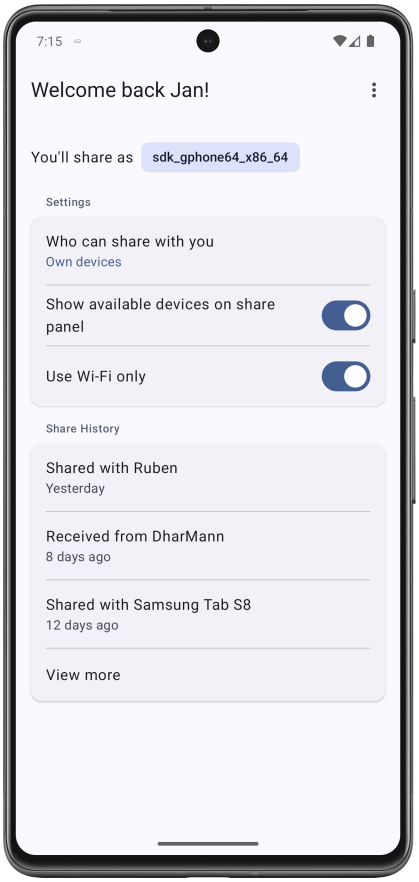
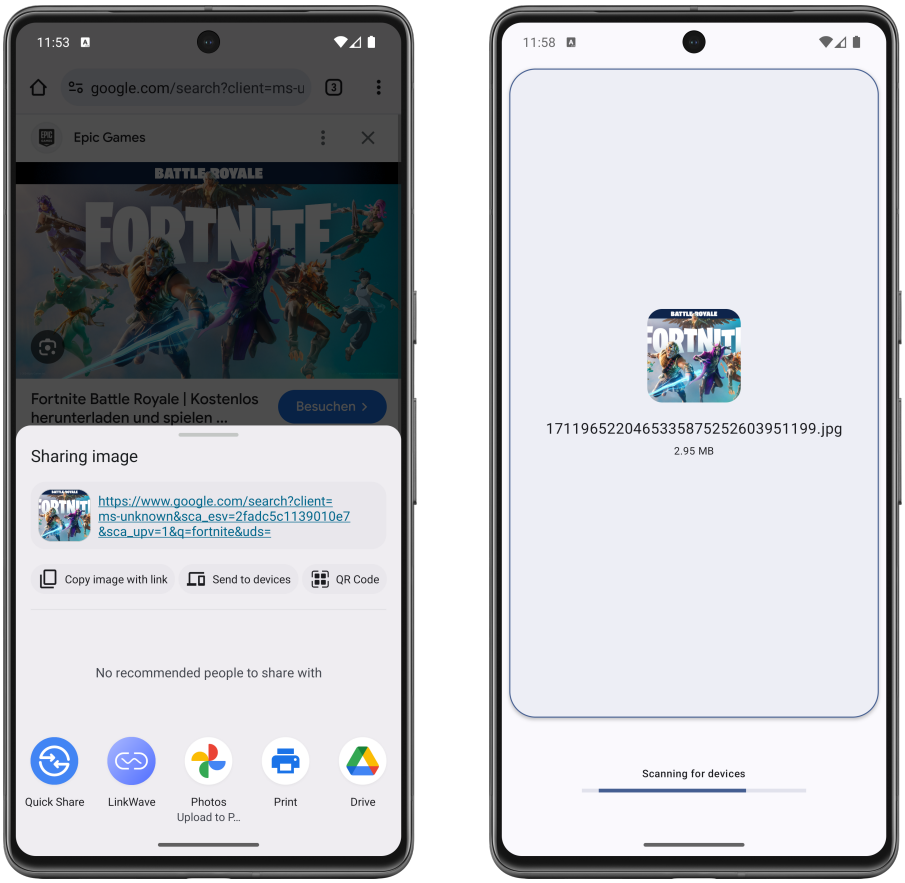


Abbildung 6: Android LinkWave App

- **Das “Teilen” Fenster**: Auch in Android gibt es die Funktion “Teilen”. Diese Funktion funktioniert beinahe identisch wie auf iOS. Bilder und Dateien können aus anderen Apps an die LinkWave App gesendet werden. Um ein Bild oder eine Datei zu teilen, muss also nur auf das Teilen-Symbol in einer App gedrückt und dann die LinkWave App ausgewählt werden. Diese Funktionsweise orientiert sich an der des Android eigenen “Quick-Share” und dem auf Apple Geräten etablierten “AirDrop”.



(a) Android Teilen Auswahl

(b) Android LinkWave Teilen Fenster

Abbildung 7: Android Teilen Erweiterung

- **Die Zwischenablage** Seit Android 10 kann eine App nur noch aus der Zwischenablage lesen, wenn sie im Vordergrund ist. Es gibt allerdings die Möglichkeit, wenn ein Text ausgewählt ist, diesen an eine App zu senden. Hierzu muss im sogenannten “Selection-Menu” auf die drei Punkte gedrückt und dann die LinkWave App ausgewählt werden. Der Text wird dann an die LinkWave App gesendet, die ihn wiederum an andere Geräte weiterleitet. In Abbildung 8 ist das Selection-Menu zu sehen.

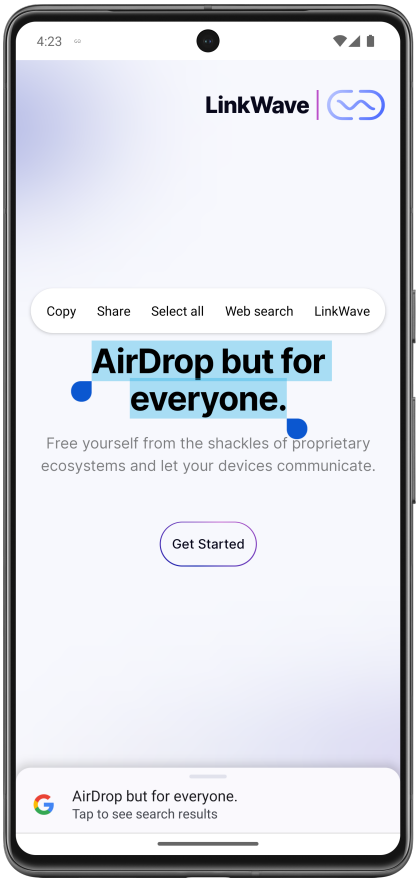


Abbildung 8: Selection-Menu

3.4 Windows

Die LinkWave App für Windows wurde in der Programmiersprache C# entwickelt. Für das UI wurde das Framework WinUI verwendet. Die App ist für Windows 10 und neuer verfügbar.

Die Windows App bietet ähnliche Möglichkeiten für die Benutzung wie die macOS App. Die implementierten Möglichkeiten sind:

- **Die LinkWave App:** In der App können Benutzer:innen genau so wie auf macOS und iOS die Dateiübertragung benutzen, sowie die Einstellungen der App ändern. Man kann sich auch mit seinem LinkWave Konto anmelden, um mehr Funktionen frei zu schalten. Genau wie unter macOS kann man auch hier eine Geräteübersicht aufrufen.

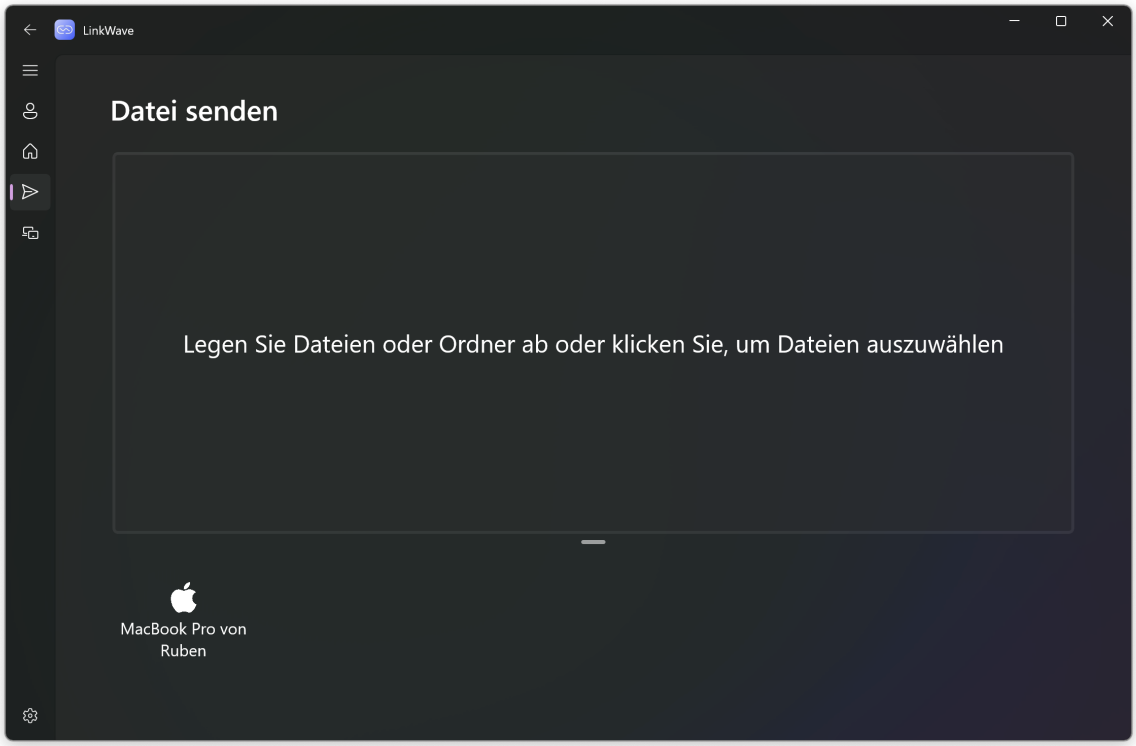


Abbildung 9: Windows LinkWave App

- **Das “Teilen” Fenster:** Auch auf Windows gibt es die Funktion “Teilen”. Diese Funktion funktioniert ähnlich wie auf macOS.

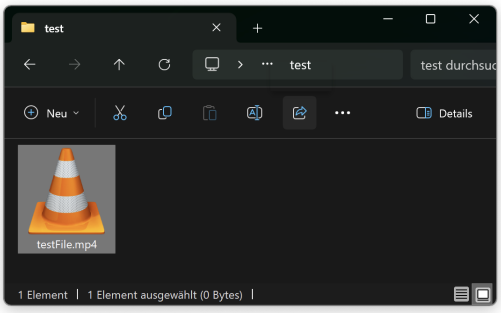
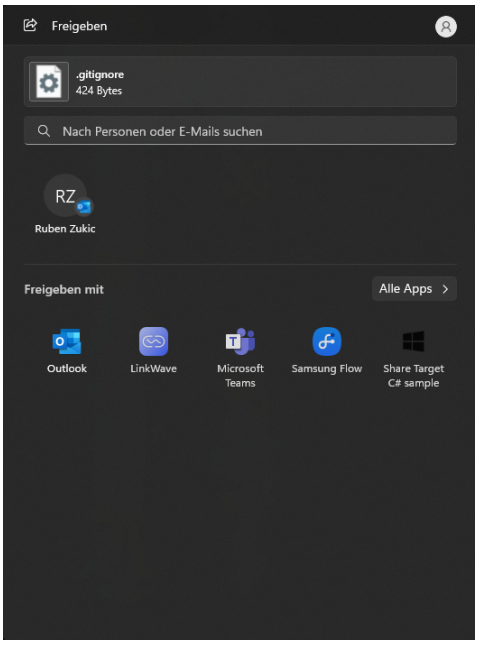


Abbildung 10: Windows Teilen in Explorer



(a) Windows Teilen Auswahl

(b) LinkWave Teilen Fenster

Abbildung 11: Windows Teilen Fenster

Man muss in einer App auf das Teilen-Symbol drücken und bekommt eine Auswahl an Apps, mit denen man den Inhalt teilen kann (siehe Abbildung 11 (a)). LinkWave ist eine dieser Apps. Wenn man LinkWave auswählt, öffnet sich das LinkWave Teilen Fenster (siehe Abbildung 11 (b)). Hier kann man auswählen, an welches Gerät man den Inhalt senden möchte.

- **Das Systemtray[2]-Icon:** Auf Windows gibt es den Systemtray. In diesem Systemtray sind die Symbole von Apps zu sehen, die im Hintergrund laufen. LinkWave hat auch ein solches Symbol (siehe Abbildung 12). Über dieses Symbol kann man die LinkWave App öffnen, sowie diese auch komplett beenden, so dass sie nicht mehr im Hintergrund läuft.

Abbildung 12: Windows Systemtray Icon

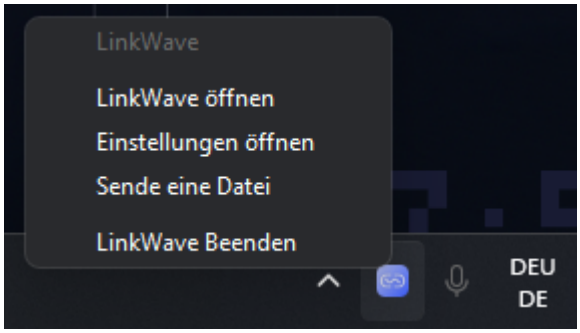


Abbildung 13: Windows Systemtray Menü

- **Die Zwischenablage:** Auf Windows gibt es ebenfalls wie auf jedem modernen Betriebssystem eine Zwischenablage. LinkWave sendet kopierte Inhalte automatisch an andere Geräte des eigenen Kontos.

[2]Das Systemtray ist ein Bereich in der Taskleiste von Windows, in dem sich Symbole von Apps befinden, die im Hintergrund laufen. Dieser ist auf der rechten Seite der Taskleiste mit dem ∧ Symbol zu finden.

3.5 Web - Dashboard

Das LinkWave Dashboard ist eine Webanwendung, die es Benutzer:innen ermöglicht, ihre Geräte und allgemeine accountbezogene Einstellungen zu verwalten. Das Dashboard wurde mit SvelteKit entwickelt und ist über den Browser auf allen Geräten verfügbar. Die Webanwendung ist für die Verwaltung des eigenen Kontos und der eigenen Geräte gedacht. Benutzer:innen können ihre eigenen Geräte sehen und bei Bedarf Löschen. Accountbezogene Einstellungen beziehen sich auf Optionen wie das Ändern des Passworts oder das Löschen des eigenen Kontos. Die Elemente der Website wurden aus der `shadcn/ui` UI-Bibliothek entnommen. Das Dashboard ist unter `account.linkwave.org` erreichbar.

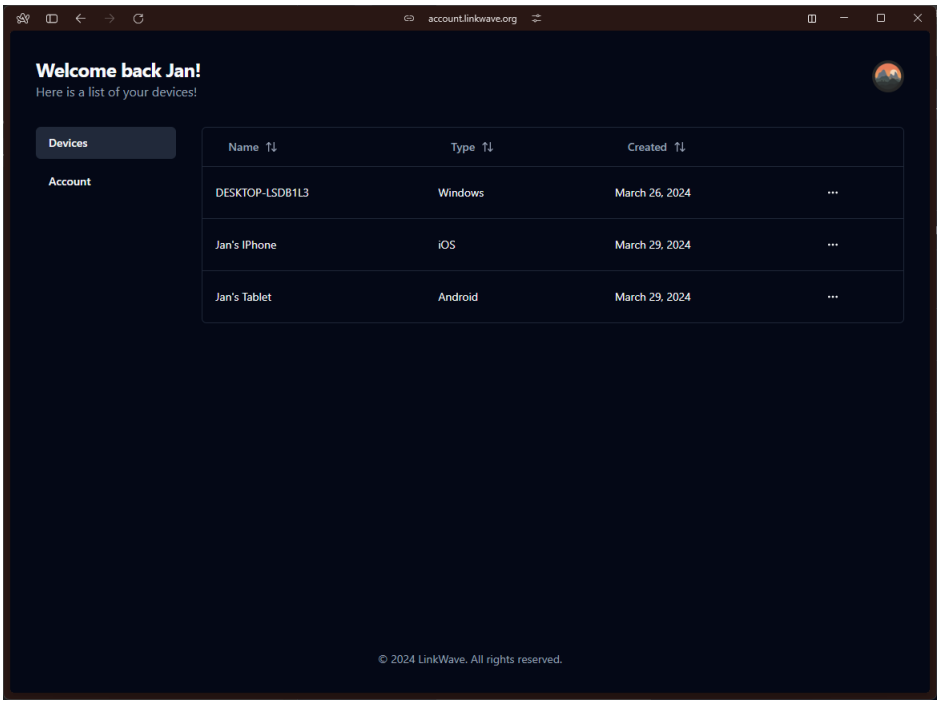


Abbildung 14: LinkWave Dashboard

4 Architektur

4.1 Ausgewählte Technologien

In diesem Abschnitt werden einige Schlüsseltechnologien und die Beweggründe für ihre Auswahl genauer erläutert.

4.1.1 WinUI

Abbildung 15: WinUI Logo

Was ist WinUI?

WinUI ist eine von Microsoft entwickelte UI Bibliothek, mit der man moderne Benutzeroberflächen für Windows-Anwendungen erstellen kann. Diese Benutzeroberflächen benutzen die selben Elemente wie die von Microsoft erstellten Systemanwendungen. [1] Diese Designsprache wird von Microsoft als “Fluent Design” bezeichnet. Die WinUI Version 3 stellt die Elemente aus den Systemanwendungen von Windows 11 zur Verfügung. Diese können aber auch von Benutzern auf älteren Windows Versionen (bis Windows 10) benutzt werden. Dort schauen Apps trotzdem genau wie unter Windows 11 aus. Die “Windows App SDK” bietet in Verwendung von C# und XAML die Möglichkeit, diese Elemente in eigenen Anwendungen zu benutzen. “Windows App SDK” wird ab Seite 26 genauer beschrieben.

Was unterscheidet WinUI von Alternativen wie MAUI?

.NET MAUI ermöglicht es ähnliche Benutzeroberflächen wie WinUI zu erstellen. MAUI unterscheidet sich jedoch in einigen Punkten von WinUI. MAUI ist eine plattformübergreifende UI Bibliothek, die es ermöglicht, Benutzeroberflächen für Windows, macOS, iOS und Android zu erstellen. WinUI hingegen ist nur für Windows verfügbar. Dadurch,

dass MAUI auf mehreren Betriebssystemen funktionieren muss, ist es jedoch schwieriger Code zu schreiben, der tief in das Betriebssystem eingreift. WinUI hingegen kann dies, da es nur für Windows gebaut wurde.

Wofür wurde WinUI verwendet?

WinUI wurde für die Entwicklung der LinkWave App für Windows verwendet. Die App wurde in C# und XAML geschrieben. Dies eignete sich gut für den Anwendungsfall von LinkWave.

4.1.2 Windows App SDK

Was ist die Windows App SDK?

“SDK” bedeutet übersetzt Software-Entwicklungs-Bausatz. Ein SDK ist eine Sammlung von Werkzeugen, die es Entwicklern ermöglicht, Software für eine bestimmte Plattform zu entwickeln. Die “Windows App SDK” ist eine SDK von Microsoft, die es ermöglicht, Apps für Windows zu entwickeln. Sie hilft moderne Windows-Apps mittels verschiedener UI Bibliotheken zu entwickeln. So könnte man beispielsweise eine App mit der “Windows App SDK” sowohl mit MAUI als auch mit WinUI entwickeln.

Die “Windows App SDK” erleichtert es auch, auf Funktionen von Windows zuzugreifen. So kann man beispielsweise leicht auf die Zwischenablage zugreifen oder Benachrichtigungen anzeigen.

Was unterscheidet die Windows App SDK von anderen SDKs wie UWP?

Die “Windows App SDK” ist eine Weiterentwicklung von UWP. UWP ist eine ältere SDK von Microsoft, die den selben Zweck erfüllt. UWP ist jedoch nur für Windows 10 verfügbar. Die “Windows App SDK” hingegen ist für Windows 10 und neuer verfügbar. Sie ermöglicht es auch, Apps für Windows 11 zu entwickeln. Leider unterstützt die “Windows App SDK” manche Funktionen, die UWP unterstützt, nicht. So muss man beispielsweise das Anmelden mit einem Google Account aufwendig selber implementierten. Trotzdem macht es Sinn sich für die “Windows App SDK” zu entscheiden, da UWP von Microsoft

schon als “veraltet” gekennzeichnet wird und daher, anders als die “Windows App SDK”, keine Erneuerungen mehr bekommt.

Wofür wurde die Windows App SDK verwendet?

Die “Windows App SDK” wurde für die Entwicklung der LinkWave App für Windows verwendet. Die App wurde in C# und XAML geschrieben. Mit der “Windows App SDK” war es möglich, die App so zu entwickeln, dass sie auf Windows 10 und neuer funktioniert und die neuesten Funktionen von Windows 11 unterstützt. Auch wurden Funktionen wie der Zugriff auf die Zwischenablage oder das Senden von Benachrichtigung durch die “Windows App SDK” deutlich erleichtert.

4.1.3 SwiftUI

Abbildung 16: SwiftUI Logo

Was ist SwiftUI?

SwiftUI ist ein von Apple entwickeltes Framework um Benutzeroberflächen für macOS (für u.a. MacBooks) und iOS (für iPhones) zu entwickeln. Man kann mit SwiftUI die selben Elemente benutzen, die auch Apple in seinen Systemanwendungen benutzt.

Was unterscheidet SwiftUI von Alternativen wie UIKit?

SwiftUI ist eine einfachere und modernere Version von UIKit. SwiftUI ist deklarativ, was bedeutet, dass man nur beschreiben muss, wie die Benutzeroberfläche aussehen soll und nicht, wie sie erstellt werden soll. Dies macht es einfacher, Benutzeroberflächen zu erstellen. Auch ist SwiftUI für Anfänger einfacher zu lernen, da es weniger Code für

die selbe Funktionalität benötigt. Ein Beispiel für den Unterschied zwischen SwiftUI und UIKit ist mit dem Code für ein simples UI Element hier zu sehen:

```
class ViewController:
  UIViewController {
  override func viewDidLoad() {
    super.viewDidLoad()
    let label = UILabel()
    label.text = "Hello, UIKit!"
    label.font = UIFont
      .systemFont(ofSize: 36)
    label.textColor = .blue
    view.addSubview(label)
  }
}
```

UIKit Beispiel

```
struct ContentView: View {
  var body: some View {
    Text("Hello, SwiftUI!")
      .font(.largeTitle)
      .foregroundColor(.blue)
  }
}
```

SwiftUI Beispiel

SwiftUI ermöglicht dank kürzerem und einfacherem Code eine schnellere Entwicklung von Benutzeroberflächen.

Wofür wurde SwiftUI verwendet?

SwiftUI wurde für die Entwicklung der LinkWave App für macOS und iOS verwendet. Die App nutzt spezielle Elemente von Apple, sodass sie aussieht und funktioniert, als wäre sie ein fester Bestandteil von macOS und iOS selbst.

4.1.4 Jetpack Compose

Abbildung 17: Jetpack Compose Logo

Jetpack Compose ist das von Google empfohlene Toolkit für die Erstellung nativer Benutzeroberflächen in Android, das den Schwerpunkt auf einen deklarativen Ansatz für die Entwicklung von Benutzeroberflächen legt. Im Gegensatz zu der traditionellen Sichtweise, XML-Layouts und Kotlin/Java-Code zu trennen, ermöglicht Compose Entwicklern, ganze UIs programmatisch mit Kotlin zu erstellen. Es vereinfacht und beschleunigt die UI-Entwicklung auf Android mit weniger Code, leistungsstarken Tools und intuitiven Kotlin-APIs. Kotlin ist eine von JetBrains entwickelte Programmiersprache, die vollständig mit Java interoperabel ist. Viele Android-Schnittstellen, einschließlich der von LinkWave genutzten Schnittstelle für das `Network Service Discovery Protokoll` (siehe Kapitel 5.1.3), werden über Java-Code bereitgestellt. Die Interoperabilität von Kotlin und Java ermöglicht es diese Schnittstellen von Jetpack Compose aus anzusprechen.

Benutzeroberfläche Anders als in WinUI oder im Web wird die Benutzeroberfläche nicht in einer Markup-Language wie `XAML` oder `HTML` definiert, sondern dynamisch im Code über sogenannte `@Composable` Funktionen generiert. Google stellt eine Bibliothek an Composables zur Verfügung, die der von Android-Systemanwendungen verwendeten Designsprache ”Material Design 3” entsprechen. Folgende Abbildung zeigt den Aufbau eines einfachen Components.

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello␣$name!")
}
```

Die Funktion `Greeting` definiert ein `Text`-Element, das den Text ”Hello” und den übergebenen Namen anzeigt.

Warum wurde Jetpack Compose ausgewählt?

Jetpack Compose wurde für die Entwicklung der LinkWave App für Android verwendet. Sie nutzt spezielle Components von Google, sodass sie aussieht und funktioniert, als wäre sie ein fester Bestandteil von Android selbst. Im Gegensatz zu anderen Technologien wie React-Native oder Flutter, die eine Abstraktionsschicht nutzen, um mit Android zu kommunizieren, spricht Jetpack Compose direkt mit Android. Das hilft, die App nahtlos in das Betriebssystem zu integrieren.

4.1.5 SvelteKit

Abbildung 18: Svelte Logo

SvelteKit ist ein modernes Framework für die Entwicklung von Webanwendungen, das auf der Svelte-Bibliothek aufbaut. SvelteKit vereinfacht den Entwicklungsprozess, indem es eine strukturierte Umgebung für die Erstellung von schnellen und reaktiven Single-Page-Applications (SPAs) oder Server-Side-Rendered (SSR) Anwendungen bietet. Im Gegensatz zu traditionellen Frameworks, bei denen die meiste Arbeit im Browser des Benutzers erfolgt, kompiliert Svelte den Code bereits beim Build-Prozess in hoch optimierten, imperative JavaScript-Code, was zu einer signifikanten Leistungssteigerung führt. SvelteKit bietet out-of-the-box Unterstützung für Routing, Datenprefetching und nahtlose Integration mit verschiedenen Backend-Systemen, was Entwicklern ermöglicht, sich auf die Entwicklung von Features statt auf Boilerplate-Code zu konzentrieren.

Wofür wurde SvelteKit verwendet?

SvelteKit wurde für die Entwicklung des LinkWave Dashboards verwendet. Das Dashboard ist eine Webanwendung, die es Benutzer:innen ermöglicht, ihre Geräte zu verwalten und allgemeine accountbezogene Einstellungen zu ändern.

4.1.6 GraphQL

GraphQL ist eine Abfragesprache für APIs, wobei eine API (Application Programming Interface) eine Schnittstelle ist, die es verschiedenen Softwareanwendungen ermöglicht, miteinander zu interagieren und Funktionen oder Daten auszutauschen. Mit GraphQL definiert der Entwickler ein Schema, das alle verfügbaren Daten und den Zugriff darauf beschreibt, sodass Clients in einer einzigen Anfrage genau spezifizieren können, welche Daten sie benötigen. Der Server interpretiert diese Anfrage, greift auf die benötigten Daten zu und gibt sie in dem durch die Anfrage definierten Format zurück. GraphQL betrachtet die abzufragenden Informationen als ein vernetztes System von nodes und edges, wodurch es möglich wird, tief verschachtelte und komplexe Abfragen zu konstruieren. Dieser Ansatz erlaubt eine zielgerichtete und effiziente Datenabfrage, die genau auf die Bedürfnisse der Clients abgestimmt ist.

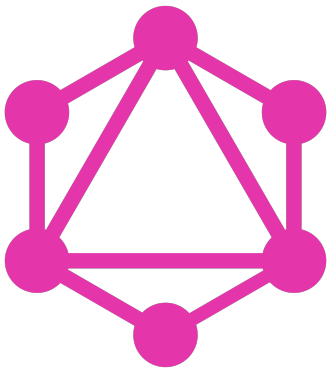


Abbildung 19: Graph QL Logo

Vorteile von GraphQL:

- **Präzise Datenanforderungen:** Ermöglicht den Benutzern, über eine einzige, einheitliche Schnittstelle genau die Daten abzufragen, die sie benötigen.
- **Stark typisiertes Schema:** Stellt eine präzise Validierung sicher und unterstützt klare Dokumentation durch ein stark typisiertes Schema.
- **Entwicklungsverbesserung:** Bietet eine Introspektionsfähigkeit, die die Entwicklung von Frontend-Anwendungen vereinfacht.
- **Minimierte Datenübertragung:** Reduziert die übertragene Datenmenge, was besonders auf mobilen Plattformen die Leistung verbessert.
- **Reduzierte Netzwerkanfragen:** Verringert die Notwendigkeit für mehrere Netzwerkanfragen und verbessert dadurch die Gesamtperformance der Anwendung.

Nachteile von GraphQL:

- **Höhere Komplexität der Abfragen:** Für neue Entwickler kann die Einarbeitung in GraphQL aufgrund seiner Komplexität und Flexibilität herausfordernder sein als REST.
- **Caching-Herausforderungen:** Das Caching auf Clientseite ist komplizierter zu implementieren, da GraphQL-Anfragen oft spezifisch und weniger vorhersehbar sind als REST-Anfragen.
- **Rate Limiting und Überwachung:** Das Überwachen und Begrenzen der Anfragerate kann aufgrund der variablen Natur der GraphQL-Anfragen schwieriger sein.
- **Performance bei großen Anfragen:** Große und komplexe Anfragen können die Performance beeinträchtigen, da sie möglicherweise viele Ressourcen auf einmal beanspruchen.

Vorteile von REST-APIs:

- **Einfachheit und Verständlichkeit:** REST APIs folgen einem standardisierten Ansatz mit klaren Konventionen, die es Entwicklern erleichtern, mit REST umzugehen.
- **Breite Unterstützung:** REST ist in vielen Plattformen und Sprachen nativ unterstützt, was die Integration in bestehende Systeme erleichtert.
- **Effizientes Caching:** REST ermöglicht ein effektives Caching von Anfragen, was die Netzwerklast und Serveranfragen reduziert.
- **Zustandslosigkeit:** Jede Anfrage von Client zu Server ist unabhängig und enthält alle Informationen, die der Server zur Beantwortung benötigt.

Nachteile von REST-APIs:

- **Überfetching und Underfetching:** REST-APIs können dazu führen, dass zu viele oder zu wenige Daten geliefert werden, was zu ineffizienten Anfragen führen kann.
- **Mehrere Endpunkte:** Die Notwendigkeit, unterschiedliche Endpunkte für verschiedene Datenressourcen zu definieren, kann die Komplexität erhöhen.
- **Starre Strukturen:** Die starren Strukturen von REST können die Entwicklung dynamischer Anwendungen erschweren, die flexible Datenanforderungen haben.
- **Schwierigkeiten bei der Aggregation von Daten:** Das Zusammenführen von Daten aus verschiedenen Quellen oder Endpunkten kann zusätzliche Anfragen und Komplexität bedeuten.

4.1.7 Bonjour

Die Einbindung von Bonjour in LinkWave erleichtert die Vernetzung auf effektive und nutzerorientierte Weise. Dank Bonjour kann LinkWave andere im Netzwerk vorhandene Geräte, die ebenfalls mit LinkWave ausgestattet sind, rasch identifizieren.

Abbildung 20: Bonjour Logo

Was ist Bonjour?

Das Bonjour-Protokoll, auch Zero Configuration Networking (Zeroconf) genannt, ist eine von Apple entwickelte Technologie. Bonjour ermöglicht es Geräten sich in einem lokalen Netzwerk zu erkennen, ohne dass eine manuelle Netzwerkkonfiguration erforderlich ist. Das Protokoll wird nicht nur von Apple-Betriebssystemen, sondern auch von vielen anderen Betriebssystemen verwendet. [17]

Wie funktioniert Bonjour?

Durch die Verwendung von Multicast DNS (mDNS) bietet Bonjour eine Plattform für Geräte, um ihre Dienste innerhalb eines lokalen Netzwerks selbstständig zu veröffentlichen und zu entdecken. Diese Technologie zeichnet sich durch eine hohe Kompatibilität über verschiedene Betriebssysteme hinweg aus und ist einer der Hauptgründe, warum LinkWave Bonjour verwendet, um andere Geräte in einem lokalen Netzwerk zu finden.

Service Discovery Das Bonjour-Protokoll nutzt **Multicast DNS (mDNS)**, eine Technik zur Erkennung von Netzwerkdiensten, die sich grundlegend vom traditionellen DNS unterscheidet. Anstelle der Abhängigkeit von zentralisierten Servern ermöglicht mDNS Geräten und Diensten, sich innerhalb des lokalen Netzwerks eigenständig zu veröffentlichen und zu finden. Dies geschieht durch das Senden von Anfragen an eine spezielle Multicast-IP-Adresse, die von allen Netzwerkgeräten gemeinsam genutzt und überwacht wird.

Namensauflösung Das Bonjour-Protokoll vereinfacht die Namensauflösung im Netzwerk. Das Merken von IP-Adressen ist nicht mehr erforderlich. Dies wird durch die Verwendung des Multicast DNS (mDNS) erreicht, das die Zuordnung von benutzerfreundlichen Namen zu IP-Adressen innerhalb des lokalen Netzwerks automatisiert. Geräte senden Multicast-Anfragen an die Adresse 224.0.0.251 für IPv4 oder ff02::fb für IPv6, um den Namen eines Geräts aufzulösen oder Dienste zu entdecken. Darüber hinaus ermöglicht dies nicht nur die Identifizierung von Diensten im Netz, sondern auch die Bereitstellung zusätzlicher Metadaten, die eine genaue Identifizierung von Diensten unterstützen.

Automatische Konfiguration Durch die Automatisierung der Netzwerkkonfiguration erleichtert Bonjour die Einrichtung von Geräten im Netzwerk erheblich. Es verwaltet die Zuweisung von IP-Adressen sowohl über DHCP als auch im APIPA-Bereich (169.254.x.x), falls kein DHCP-Server verfügbar ist.

Breite Kompatibilität Obwohl von Apple entwickelt, unterstützt Bonjour eine Vielzahl von Betriebssystemen über macOS und iOS hinaus, darunter Windows (mit installierter Bonjour-Software) sowie andere Systeme, die mDNS und DNS-SD implementieren. Dies unterstreicht seine plattformübergreifende Anwendbarkeit.

Zero-Configuration Networking (Zeroconf) Das Konzept der "Null-Konfiguration" impliziert, dass Geräte mit minimalem bis gar keinem Eingriff von Seiten des Benutzers funktionieren. Dies ist insbesondere für Benutzer ohne technische Vorkenntnisse oder jene, die keine komplizierten Einstellungen vornehmen möchten, von Vorteil.

4.2 Netzwerkarchitektur

LinkWave verwendet ein Client-Server-Modell, um Anfragen zu empfangen und zu bearbeiten. Die Kommunikation erfolgt über das TCP/IP-Protokoll, das eine sichere und zuverlässige Datenübertragung gewährleistet. Der Server übernimmt dabei die Rolle des Empfängers und verarbeitet die von den Clients gesendeten Datenanfragen. Die Sicherheit der Daten wird durch das SSL/TLS-Protokoll gewährleistet, das eine Verschlüsselung und Authentifizierung der kommunizierenden Parteien ermöglicht. Zu Beginn jeder Verbindung findet ein Handshake statt, bei dem Zertifikate und kryptographische Schlüssel ausgetauscht werden, um eine sichere Verbindung herzustellen. Eingehende Client-Anfragen nimmt der Server über TCP-Listener entgegen und leitet sie intern an die entsprechenden Dienste weiter, die für verschiedene Aufgaben wie File- oder Clipboard-Sharing zuständig sind. Dieser Prozess wird in den folgenden Abschnitten erläutert.

4.3 Softwarearchitektur

4.3.1 iOS

Im folgenden Abschnitt wird die Softwarearchitektur der iOS App von LinkWave beschrieben. Die iOS App wurde in Swift mit SwiftUI geschrieben und arbeitet wie die Windows App auch (siehe Kapitel 4.3.4) nach dem MVVM (Model-View-ViewModel) Architekturmuster.

iOS Grundkonzepte

Jede iOS App besteht aus mehreren Erweiterungen (Extensions), die zusammenarbeiten, um die Funktionalität der App bereitzustellen. Jede Erweiterung hat eine spezifische Aufgabe und ist unabhängig von den anderen. Jede iOS App hat zumindest eine Hauptanwendung, die die Benutzeroberfläche der App (siehe Kapitel 3.1) an sich steuert. Andere Erweiterungen können entweder Code logisch gruppieren oder Funktionen bereitstellen, die auch ohne die Hauptanwendung funktionieren. Verwendet wurden in der iOS App von LinkWave folgende Erweiterungen:

- **App (LinkWaveApp):** Die Hauptanwendung, die die Benutzeroberfläche der App steuert. Die genaue Implementierung der App wird in Kapitel 3.1 beschrieben.
- **Framework (LinkWaveFramework):** Eine Sammlung von Klassen und Funktionen, die von der Hauptanwendung logisch getrennt sind. Frameworks erfüllen den selben Zweck wie Klassenbibliotheken in anderen Programmiersprachen wie C#. Sie ermöglichen es, dass andere Erweiterungen, die nicht auf Code der Hauptanwendung zugreifen können, auf Funktionen und Klassen zugreifen können, die von sowohl der Hauptanwendung als auch anderen Erweiterungen verwendet werden. Dieses Framework wird sich mit der macOS App geteilt.
- **Share Extension (LinkWaveShare):** Eine Erweiterung, die es ermöglicht, Inhalte aus anderen Apps in die LinkWave App zu teilen. Zum Beispiel können Benutzer:innen eine Datei aus der Dateien App in die LinkWave App teilen, um die Datei an ein anderes Gerät zu senden. Diese Erweiterung ist unabhängig von der

Hauptanwendung und kann auch ohne die Hauptanwendung funktionieren. Sie wird trotzdem mit der Hauptanwendung installiert und kann auf Funktionen und Klassen des Frameworks zugreifen.

LinkWaveApp im Detail Aus der `LinkWaveApp` lässt sich eine App erstellen, die im App Store veröffentlicht werden kann. Die Hauptanwendung der iOS App von LinkWave ist die LinkWaveApp. Sie besteht aus mehreren Views, die in SwiftUI geschrieben sind. Jede View stellt eine Bildschirmseite der App dar, die die Benutzer:innen sehen und mit der sie interagieren können. Das bedeuted, dass das komplette UI (User Interface) der App in der LinkWaveApp definiert ist.

LinkWaveFramework im Detail Das `LinkWaveFramework` enthält Klassen und Funktionen, die von der LinkWaveApp und von der LinkWaveShare Erweiterung benutzt werden. Es enthält die Logik, die für die Kommunikation mit anderen Geräten im Netzwerk und die Verwaltung von Dateien und der Zwischenablage benötigt wird. Das Framework ist in Swift geschrieben.

LinkWaveShare im Detail Die `LinkWaveShare` Erweiterung ermöglicht es, Inhalte aus anderen Apps in die LinkWave App zu teilen. Sie besteht aus einer View, die es den Benutzer:innen ermöglicht, den Inhalt zu teilen, und einer Klasse, die die Logik für das Teilen des Inhalts enthält. Die Erweiterung ist in Swift geschrieben und verwendet das LinkWaveFramework, um die Kommunikation mit anderen Geräten zu ermöglichen. Die Erweiterung wird in Kapitel 3.1 genauer beschrieben.

4.3.2 macOS

Im folgenden Abschnitt wird die Softwarearchitektur der macOS App von LinkWave beschrieben. Die macOS App wurde in Swift mit SwiftUI geschrieben und arbeitet wie die Windows App auch (siehe Kapitel 4.3.4) nach dem MVVM (Model-View-ViewModel) Architekturmuster.

macOS Grundkonzepte

SwiftUI Apps auf macOS funktionieren genau gleich wie SwiftUI Apps auf iOS. Der Hauptunterschied ist, dass macOS Apps auf einem Mac laufen und iOS Apps auf einem iPhone oder iPad. Die Architektur der macOS App von LinkWave ist identisch zur iOS App, mit der Ausnahme, dass die Benutzeroberfläche an macOS angepasst ist.

Zudem lassen sich macOS Apps deutlich leichter als iOS Apps veröffentlichen. Während iOS Apps nur im Apple App Store veröffentlicht werden können, können macOS Apps auch über einen Download-Link in Form einer .dmg Datei veröffentlicht werden. Dies ermöglicht es, macOS Apps auch ohne den App Store zu verteilen.

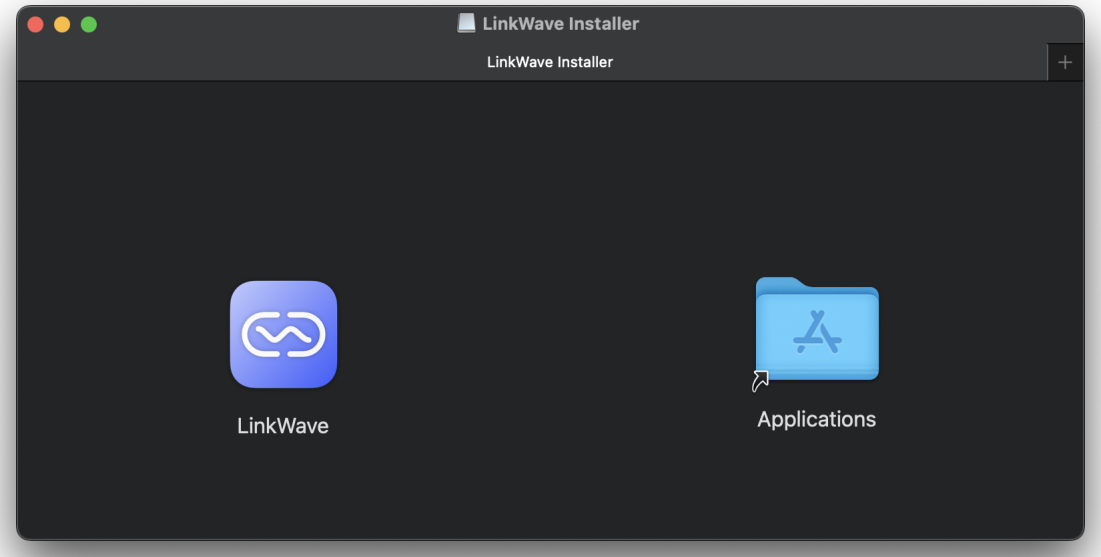


Abbildung 21: MacOS .dmg Installer

Die MacOS App von LinkWave wurde in Swift mit SwiftUI geschrieben und ist genau gleich aufgebaut wie die iOS App (siehe in Kapitel 4.3.1). Sie besteht aus den drei Komponenten:

- `LinkWaveApp`: Für die Benutzeroberfläche der App zuständig.
- `LinkWaveFramework`: Enthält Logik die sich App und Share Extension teilen.
- `LinkWaveShare`: Ermöglicht es, Inhalte aus anderen Apps in die LinkWave App zu teilen.

Diese haben im Grunde die selbe Funktionalität wie in der iOS App. Unter macOS wurde jedoch die `LinkWaveApp` angepasst.

LinkWaveApp im Detail Die `LinkWaveApp` ist für die Benutzeroberfläche der App zuständig. Was die `LinkWaveApp` auf macOS von der auf iOS unterscheidet, ist, dass macOS nicht nur ein Fenster hat und zudem andere Funktionen wie ein Icon in der Menüleiste unterstützt. Dieses befinden sich in der `LinkWaveApp`, da sie nur verfügbar sein sollen, wenn die App geöffnet ist.

Das bedeutet, dass neben der View für das Hauptfenster der App auch Views für das Icon in der Menüleiste und das Fenster für die Einstellungen der App in der `LinkWaveApp` definiert sind. In der `LinkWaveApp` sind folgende Views vorhanden:

- **Window(id: “main”)**: Das Hauptfenster der App (siehe Kapitel 3.2). Dieses Fenster beinhaltet eine `NavigationSpliView` mit einer variablen `DetailView`, die je nach Seitenauswahl des Benutzers wechselt. Diese View ermöglicht es, der App eine Seitenleiste mit verschiedenen Seiten zu geben.
- **Window(id: “permissions”)**: Ein Fenster, das dem Benutzer anzeigt, welche Berechtigungen die App benötigt.
- **MenubarExtra**: Das Icon in der Menüleiste, das dem Benutzer ermöglicht, die App zu öffnen und zu schließen.
- **Settings**: Ein Fenster, in dem der Benutzer die Einstellungen der App ändern kann.

4.3.3 Android

Im folgenden Abschnitt wird die Softwarearchitektur der Android App von LinkWave beschrieben. Die Android App wurde in Kotlin mit Jetpack Compose geschrieben und arbeitet wie die Windows App (siehe Kapitel 4.3.4) auch nach dem MVVM (Model-View-ViewModel) Architekturmuster.

Android Grundkonzepte

Jede Android App besteht aus mehreren Komponenten, die zusammenarbeiten, um die Funktionalität der App bereitzustellen. Die wichtigsten Komponenten werden im Folgenden kurz vorgestellt.

Activities Eine Activity kann man als eine Bildschirmseite einer App verstehen. Jeder Bildschirm, mit dem man in einer App interagiert (z. B. Anmeldebildschirm, Startbildschirm, Einstellungsbildschirm), wird in der Regel von einer eigenen Activity gesteuert. Eine Activity stellt das Fenster dar, in dem die App das UI (User Interface) zeichnet. Jede Activity ist unabhängig von den anderen und kann andere Activities über sogenannte Intents starten. Folgende Abbildung zeigt ein Beispiel für eine Activity in Kotlin:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MaterialTheme {
                Greeting("Android")
            }
        }
    }
}
```

Intents Intents sind Messaging-Objekte, die verwendet werden, um eine Aktion von einer anderen App-Komponente anzufordern. Das kann das Starten einer anderen Activity oder eines Services sein oder das Nutzen von Android-Funktionen wie das Öffnen

einer Webseite. Intents können auch dazu verwendet werden, Daten zwischen Komponenten zu übermitteln. Folgende Abbildung zeigt einen Intent, der die YouTube-App öffnet:

```
val intent = Intent(Intent.ACTION_MAIN).apply {
    `package` = "com.google.android.youtube"
}
startActivity(intent)
```

Hier wird ein `MAIN`-Intent erstellt, der an das Package `com.google.android.youtube` gesendet wird. Es gibt eine Vielzahl von Intents, die von Android bereitgestellt werden, um verschiedene Aktionen auszuführen. Es kann zum Beispiel mit einem `SEND`-Intent eine Datei an eine andere App gesendet werden. Der `MAIN`-Intent startet die `MainActivity` einer App. Welche Intents eine App unterstützt, wird in der `AndroidManifest.xml` Datei festgelegt.

Services Ein Service ist eine Komponente, die im Hintergrund ausgeführt wird, um lang laufende Operationen durchzuführen. Ein Service kann beispielsweise im Hintergrund Musik abspielen, ohne dass der Benutzer die App im Vordergrund hat. LinkWave verwendet einen Service um das Gerät im Hintergrund im Netzwerk bekannt zu machen. Folgende Abbildung zeigt ein Beispiel für einen Service:

```
class LinkWaveService : Service() {
    override fun onStartCommand(
        intent: Intent,
        flags: Int, startId: Int
    ): Int {
        startForeground(1, notification)
        startDiscovery()
        return super.onStartCommand(intent, flags, startId)
    }
}
```

Seit Android API Level 26 (Android 8.0) müssen Services im Vordergrund laufen. Das heißt sie müssen eine Benachrichtigung anzeigen, die den Benutzer informiert, dass

der Service im Hintergrund läuft und Ressourcen verbraucht. Solche Services werden als `Foreground Services` bezeichnet.

Broadcast Receiver Broadcast Receiver ermöglichen der App, Broadcasts von anderen Apps oder dem System zu empfangen. Ein Beispiel wäre das Hören auf einen Broadcast, der anzeigt, dass der Akku des Geräts fast leer ist oder dass das Gerät neu gestartet wurde.

```
class BootCompletedReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        if (Intent.ACTION_BOOT_COMPLETED == intent.action) {
            // Code to execute when the boot is completed
        }
    }
}
```

Content Provider Content Provider ermöglichen es, Daten zwischen Apps zu teilen. Ein Content Provider stellt eine standardisierte Schnittstelle zur Verfügung, um auf Daten zuzugreifen. Ein Beispiel wäre das Teilen von Kontakten zwischen verschiedenen Apps oder das Auflisten aller Media-Dateien, also Bilder und Videos, auf dem Gerät.

AndroidManifest.xml Das AndroidManifest.xml ist die Konfigurationsdatei in der die App-Komponenten konfiguriert werden. Es wird festgelegt, welche Komponenten es gibt, welche Berechtigungen die App benötigt und welche Intents von der App unterstützt werden. Hier wird auch festgelegt, welche Activity beim Start der App geöffnet wird.

Trailing Lambdas in Kotlin In Kotlin gibt es die Möglichkeit, Funktionen als letztes Argument einer Funktion zu übergeben. Dies wird als Trailing Lambda bezeichnet. Wenn ein Lambda-Ausdruck der letzte Parameter einer Funktion ist, kann er außerhalb der runden Klammern der Funktionsaufrufe platziert werden. In Jetpack Compose wird diese Syntax häufig verwendet, um den UI-Code strukturierter und deklarativer zu gestalten.

```
@Composable
fun FancyBox(content: @Composable () -> Unit) {
    Box {
        content()
    }
}
```

Das Beispiel zeigt eine Funktion `FancyBox`, die ein `Box`-Element mit einem beliebigen Inhalt erstellt. Der Inhalt wird als Lambda-Ausdruck übergeben. Dieser Komponent kann dann wie folgt verwendet werden:

```
FancyBox {
    Button(onClick = { /*TODO*/ }) {
        Text(text = "Click␣me!")
    }
}
```

`Button` ist eine weitere Composable-Funktion, die zwei Parameter erwartet: `onClick` und `content`. `onClick` ist eine Funktion, die ausgeführt wird, wenn der Button geklickt wird. `content` ist der Inhalt des Buttons, der in diesem Fall ein Text-Element ist.

Dependency Injection Dependency Injection (DI) ist ein Entwurfsmuster, das es einem Programm ermöglicht, unabhängig davon zu sein, wie seine Dependencies bzw. Abhängigkeiten erstellt, zusammengesetzt und repräsentiert werden. Es geht darum, die fest codierten Abhängigkeiten zu entfernen und die Möglichkeit zu schaffen, diese entweder zur Laufzeit oder zur Kompilierzeit zu ändern. Das ermöglicht es, bei Tests Mock-Objekte zu verwenden, um die Abhängigkeiten zu ersetzen. Für die Entwicklung der LinkWave App wurde das DI-Framework Koin verwendet.

```
class DevicesViewModel(
    private val bonjourController: BonjourController
) : ViewModel() { ... }
```

Dieses ViewModel ist abhängig von einem `BonjourController`. Dieser wird im Konstruktor übergeben. Koin sorgt dafür, dass der `BonjourController` zur Laufzeit erstellt und dem ViewModel übergeben wird.

```
val appModule = module {
    single { BonjourController() }
    viewModel { DevicesViewModel(get()) }
}
```

`get()` gibt den BonjourController zurück, der von Koin zur Laufzeit erstellt wird. `single` bedeutet, dass der BonjourController nur einmal erstellt wird und bei jedem weiteren Aufruf von `get()` der gleiche BonjourController zurückgegeben wird.

Android Programmstruktur

Wie oben schon erwähnt wurde arbeitet die Android App mit dem MVVM-Architekturmuster. Wie die verchiedenen Schichten des MVVM-Musters zusammenhängen wird in Abbildung 22 gezeigt. Der OS-Layer (Operating System Layer) startet über einen Intent eine bestimmte Activity. Die Activity greift über ViewModels, die den Presentation-Layer bilden auf Daten des Data-Layer zu.

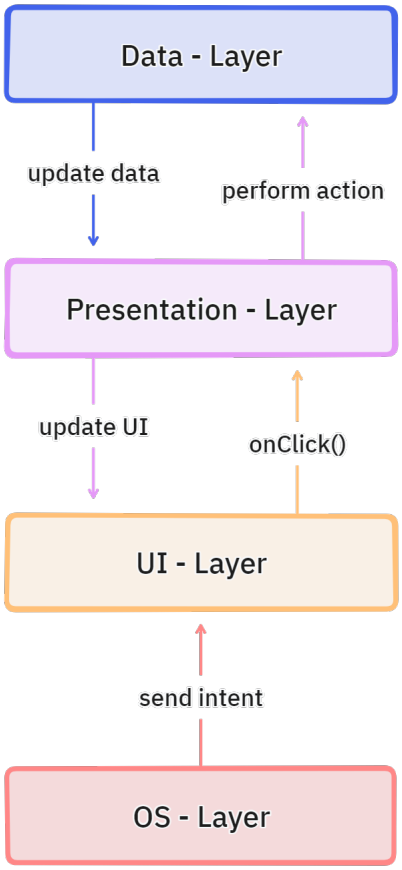


Abbildung 22: OS - Features

Activities Die Activities sind Teil des UI-Layers. Die App besteht aus drei Activities:

- **MainActivity:** Die `MainActivity` ist das Fenster, das sich öffnet, wenn auf das Icon der App am Startbildschirm von Android, dem sogenannten Launcher, gedrückt wird. Der Launcher sendet dann einen `MAIN`-Intent and die App, die dann

die `MainActivity` öffnet. In der `MainActivity` können Einstellungen wie die Sichtbarkeit im Netzwerk eingestellt werden und der Benutzende kann sich aus- und einloggen.

- **ShareActivity** Die `ShareActivity` wird geöffnet, wenn ein `SEND`-Intent an die App gesendet wird. Das passiert, wenn ein User die “Teilen”-Funktion von Android benutzt, um eine Datei zu teilen. In der `ShareActivity` kann der User dann ein Gerät auswählen, an das die Datei gesendet werden soll. Das ausgewählte Gerät wird dann über einen `FileSendManager` an das Gerät gesendet. Der `FileSendManager` stellt Informationen über den Status des Transfers bereit, die über ein ViewModel an den UI-Layer weitergegeben werden.

- **ClipboardActivity** Die `ClipboardActivity` wird geöffnet, wenn ein `PROCESS_TEXT`-Intent an die App gesendet wird. Anders als die `ShareActivity` und auch die `MainActivity` zeigt die ClipboardActivity kein UI an. Sie gibt lediglich den Ausgewählten Text über ein ViewModel an den `ClipboardShareManager` weiter. Der `ClipboardShareManager` sendet dann den Text an alle Geräte, die zum gleichen Account gehören, weiter.

Daten Auf die für die jeweilige Activity benötigten Daten kann über ViewModels zugegriffen werden. Die ViewModels sind Teil des Presentation-Layers der App. Sie enthalten die Logik, die benötigt wird, um die Daten aus dem Data-Layer zu verarbeiten und an die UI-Komponenten weiterzugeben. Um zum Beispiel die Liste der gefundenen Geräte in der `MainActivity` anzuzeigen, wird das `DevicesViewModel` verwendet. Das `DevicesViewModel` enthält eine Liste von Geräten, die über den `BonjourController` gefunden wurden. Diese Liste wird dann an die `MainActivity` übergeben, die sie dann anzeigt. Dieser Datenstrom wird über Kotlin Flow [6] realisiert.

LinkWaveService Da einige Funktionen im Hintergrund verfügbar sein sollen, wird ein Foreground-Service benötigt. Auch wenn die App im Hintergrund oder geschlossen ist, soll das Gerät im Netzwerk erkennbar sein und Anfragen wie das Teilen einer Datei annehmen können. Der `LinkWaveService` ist ein Foreground-Service der automatisch startet, wenn das Gerät neu gestartet oder die App geöffnet wird. Der Service instanziert

und startet den `BonjourAdvertiser`, der dafür sorgt, dass das Gerät im Netzwerk gefunden wird und einen `RequestListener`, der TCP-Anfragen wie Dateiübertragungen oder das Teilen der Zwischenablage entgegennimmt. So kann das Gerät Anfragen entgegennehmen, auch wenn die App geschlossen ist. In Abbildung 23 ist das Klassendiagramm des LinkWaveService zu sehen.

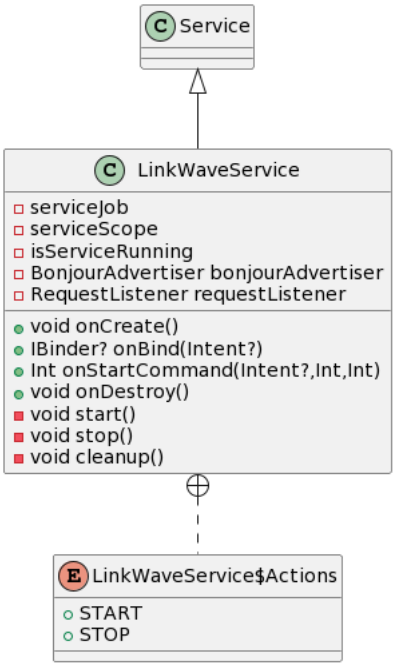


Abbildung 23: UML von LinkWaveService

4.3.4 Windows

MVVM

In diesem Abschnitt wird beschrieben wie die Softwarearchitektur der Windows App von LinkWave aufgebaut ist. Die Windows App wurde in C# und XAML geschrieben. Die App wurde in MVVM (Model-View-ViewModel) Architektur geschrieben. MVVM ist ein Entwurfsmuster, das es ermöglicht, die Benutzeroberfläche von der Logik zu trennen. Dies ermöglicht es, die Logik leichter zu testen und zu warten. Die MVVM Architektur besteht aus drei Teilen:

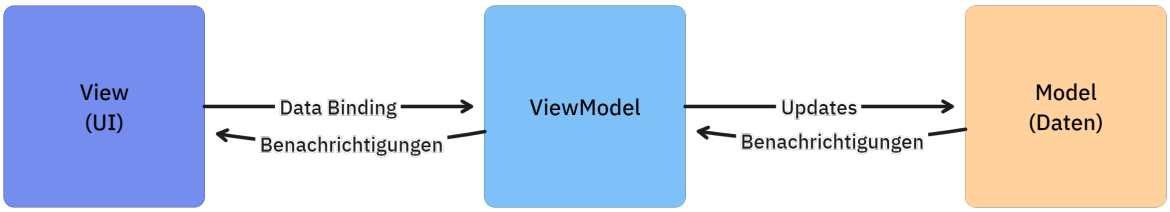


Abbildung 24: MVVM Architektur

- **Model:** Das Model ist die Datenstruktur, die die Daten der App speichert. Das Model enthält keine Logik, sondern nur die Daten. In der Windows App von LinkWave enthält das Model beispielsweise die Daten der Benutzer:innen und die Daten der Geräte.
- **View:** Die View ist die Benutzeroberfläche der App. Die View zeigt die Daten des Models an und leitet die Benutzereingaben an das ViewModel weiter. In der Windows App von LinkWave ist die View beispielsweise das Fenster, in dem die Benutzer:innen die Dateiübertragung starten können.
- **ViewModel:** Das ViewModel enhält die Logik der App, die die Daten des Models verarbeitet und an die View weitergibt. Das ViewModel enthält auch die Logik, die die Benutzereingaben verarbeitet und an das Model weitergibt. In der Windows App von LinkWave enthält das ViewModel beispielsweise die Logik, die die Dateiübertragung startet.

Windows C# Projektstruktur

Microsoft empfiehlt WinUI 3 Projekte mit den Visual Studio Erweiterung “Template Studio for WinUI (C#)” zu erstellen. Diese Erweiterung erstellt ein Projekt mit einer vorgefertigten Projektstruktur. Diese Projektstruktur besteht aus den folgenden Komponenten:

- **Das UI Projekt:** Das UI Projekt ist eine WinUI Applikation und enthält die Benutzeroberfläche der App. Aus diesem Projekt wird die App erstellt. Alle anderen Projekte beinhalten nur Code, der von diesem Projekt benutzt wird.
- **Das Core Projekt:** Das Core Projekt ist eine Klassenbibliothek und enthält die Kernlogik der App. Hier sind Hilfsfunktionen, die von der Benutzeroberfläche benutzt werden, enthalten. Diese Erleichtern die Benutzung von Funktionen des Backend Projekts. Im Core Projekt an sich sind keine Funktionen enthalten, die direkt auf Netzwerk oder Dateisystem zugreifen.
- **Das Backend Projekt:** Das Backend Projekt ist eine Klassenbibliothek und enthält die Logik, die direkt auf Netzwerk oder Dateisystem zugreift. Hier sind Funktionen enthalten, die beispielsweise Dateien übertragen oder Geräte erkennen. Das Backend Projekt wird vom Core Projekt benutzt, um die Kernlogik der App zu implementieren.

UI Projekt im Detail

Aus dem UI Projekt lässt sich eine “Packaged App” erstellen. Diese App kann in Form von einer `.msix` Datei verbreitet und auf jedem Windows ab Windows 10 installiert werden. Die Installation erfolgt über das Doppelklicken auf die Datei und schaut wie folgt aus:

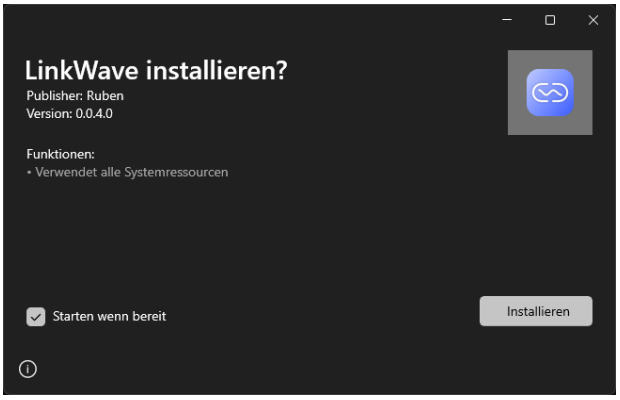


Abbildung 25: Packaged App Installation

Dadruch wird die App mit all ihren Komponenten auf dem Gerät installiert. Das bedeutet, dass sowohl die App an sich, als auch die “Teilen Erweiterung” installiert werden.

Die App an sich besteht aus einem Fenster (`MainWindow.xaml`). In diesem Fenster befindet sich eine Seite auf der das Menü und ein Rahmen für die verschiedenen Seiten der App ist (`ShellPage.xaml`). Die verschiedenen Seiten sind in Form von WinUI Pages (`<name>.xaml`) implementiert, die in der ShellPage angezeigt werden. Die Seiten sind:

- **MainPage.xaml:** Die Hauptseite der App. Wenn man die App öffnet, wird diese angezeigt. Hier findet man einen Überblick über die verschiedenen Funktionen der App, sowie über sein Konto.
- **AccountPage.xaml:** Die Accountseite der App. Hier kann man sich mit seinem Konto anmelden und Einstellungen zu seinem Konto ändern.
- **DevicesPage.xaml:** Die Geräteseite der App. Hier sieht man eine Übersicht über die LinkWave benutzenden Geräte in der Umgebung.

- **SendFilePage.xaml:** Die Dateiübertragungsseite der App. Hier kann man Dateien an andere Geräte senden.
- **SettingPage.xaml:** Die Einstellungsseite der App. Hier kann man allgemeine Einstellungen der App ändern.

Die “Teilen Erweiterung” ist in Form eines Freigabeziels umgesetzt und hat ein eigenes Fenster (`ShareWindow.xaml`). In diesem Fenster wird eine einzige Seite dargestellt (`SharePage.xaml`).

Core Projekt im Detail

Das Core Projekt enthält die Kernlogik der App. Hier sind Hilfsfunktionen, die von der Benutzeroberfläche benutzt werden, enthalten. Diese erleichtern die Benutzung von Funktionen des Backend Projekts. Diese Funktionen sind in Hilfsklassen (Services) organisiert. Die wichtigsten Klassen sind:

- **LinkWaveService:** Der LinkWaveService verwaltet jegliche LinkWave Funktionen wie beispielsweise das Finden von Geräten in der Nähe. Der LinkWaveService kümmert sich um alles, was die Hauptapp mit dem Backend Projekt macht.
- **LinkWaveShareService:** Der LinkWaveShareService verwaltet jegliche LinkWave Funktionen, die mit der Teilen Erweiterung zu tun haben. Es erfüllt ähnliche Aufgaben wie der LinkWaveService, jedoch nur wenn diese von der Teilen Erweiterung gebraucht werden. So kann der LinkWaveShareService beispielsweise keine Dateien von anderen Geräte empfangen.

Backend Projekt im Detail

Das Backend Projekt enthält die Logik, die direkt auf Netzwerk oder Dateisystem zugreift. Hier sind Funktionen enthalten, die beispielsweise Dateien übertragen oder Geräte erkennen. Das Backend Projekt wird vom Core Projekt benutzt, um die Kernlogik der App zu implementieren.

Die LinkWave.Backend DLL spielt eine wichtige Rolle im Gesamtsystem und ermöglicht die Implementierung wesentlicher Dienste wie FileSharing, ClipboardSharing, sichere Verschlüsselung und effizientes Caching von Daten.

Kernkomponenten des LinkWave.Backend DLL

Der LinkWave Receiver In der LinkWave.Backend Architektur gibt es die Komponente `LinkWaveReceiver.cs`, die als primärer Empfänger für alle eingehenden Anfragen dient. Diese zentrale Schnittstelle nimmt die Anfragen entgegen und leitet sie an RequestSwitcher.cs weiter, um die entsprechenden Funktionen aufzurufen.

Der RequestSwitcher Eine weitere Schlüsselkomponente ist der `RequestSwitcher.cs`, der eine wichtige Rolle bei der Weiterleitung von Anfragen an bestimmte Funktionen spielt. Er fungiert als Dispatcher, der jede Anfrage analysiert und je nach Inhalt und Zweck an die zuständige Funktion weiterleitet.

Durch den Einsatz des RequestSwitchers kann LinkWave.Backend gezielt auf unterschiedliche Anforderungen reagieren, indem Anfragen effizient an die dafür vorgesehenen Verarbeitungseinheiten weitergeleitet werden.

SSL Handshake Die `SSL.cs` ist zentral für die Implementierung des SSL-Handshakes, der eine sichere und verschlüsselte Verbindung zwischen Client und Server herstellt. Diese Klasse stellt sicher, dass alle Datenübertragungen vor unberechtigtem Zugriff geschützt sind.

Chaching `SqliteHandler.cs` fungiert als Schnittstelle zur internen SQLite-Datenbank und bietet Wrapper-Funktionen für Datenabfragen sowie das Hinzufügen und Aktualisieren von Daten. Diese Klasse spielt eine wesentliche Rolle bei der Verwaltung von Cache-Informationen und gewährleistet eine effiziente Datenverarbeitung innerhalb der Anwendung.

File und Clipboard Sender/Receiver Die Klassen `FileSender.cs`, `FileReceiver.cs`, `ClipboardSender.cs` und `ClipboardReceiver.cs` sind speziell für das Senden und Empfangen des Inhalts von Dateien oder der Zwischenablage konzipiert. Diese Klassen ermöglichen einen nahtlosen Datenaustausch zwischen Benutzern.

4.3.5 Server

Überblick

Der LinkWave Backend Service ist als zentrale Schnittstelle konzipiert und spielt eine entscheidende Rolle bei der Verwaltung und Sicherheit des Systems. Seine Hauptaufgabe ist die Authentifizierung von Benutzern und Geräten, um sicherzustellen, dass nur autorisierte Akteur:innen Zugang erhalten. Darüber hinaus ist er für die Speicherung der Daten verantwortlich und gewährleistet hohe Standards bzw. Best Practices für Datenintegrität und Datenschutz.
LinkWave verwendet GraphQL als Abfragesprache für die Kommunikation mit seiner API. Durch spezielle Sicherheitsmaßnahmen in GraphQL bleiben kritische Daten geschützt und werden ausschließlich vom Server verwaltet. Für den Schutz von Passwörtern setzt LinkWave auf einen besonders robusten Algorithmus namens "Bcrypt". Im Vergleich zu anderen häufig verwendeten Hashfunktionen, wie z.B. SHA-256, bietet Bcrypt einen höheren Schutz gegen Brute-Force-Angriffe, indem es diese durch seinen rechenintensiven Algorithmus deutlich verlangsamt. Damit wird ein wesentlicher Beitrag zum Schutz der Nutzerdaten geleistet. Zur Speicherung der Daten verwendet LinkWave das für seine Leistungsfähigkeit bekannte Datenbanksystem PostgreSQL. Das Design der Datenbank folgt dem Code-First-Prinzip unter Verwendung des EF Core .NET Frameworks. Dies ermöglicht eine flexible Datenstrukturentwicklung und erleichtert, die Datenbank zu verwalten und zu erweitern.

Konzept

In der Backend Architektur von LinkWave stellen wir einen effizienten Kommunikationskanal zur Verfügung. Dieser ist über mehrere Plattformen hinweg konsistent. Die Interaktion beginnt, wenn die Website, Desktop-App oder Mobile-App des Clients eine Anfrage sendet. Diese Anfrage wird dann zum Server weitergeleitet, die diese durch die API verarbeitet.

Ein wesentlicher Bestandteil unseres Servers ist die GraphQL-API, die für die Verarbeitung spezifischer Abfragen entwickelt wurde. Zur Implementierung von GraphQL wurde Hot Chocolate verwendet, eine Bibliothek, die die Implementierung von GraphQL in .NET-Anwendungen erleichtert [9]. Hot Chocolate bietet eine Vielzahl von Funktionen, die die Entwicklung von GraphQL-APIs beschleunigen und die Integration in bestehende .NET-Anwendungen erleichtern. Dies bedeutet, dass die App-Nnutzer genau die Informationen abfragen können, die sie benötigen, wodurch die Übertragung überflüssiger Daten reduziert und die Systemleistung optimiert wird.

Sobald eine Anfrage von der GraphQL-API verarbeitet wurde, erfolgt die Interaktion mit der PostgreSQL-Datenbank. Hier werden CRUD-Operationen - also das Erstellen (Create), Lesen (Read), Aktualisieren (Update) und Löschen (Delete) von Daten - ausgeführt. Aufgrund seiner Zuverlässigkeit wurde PostgreSQL für die Speicherung und Abfrage von Daten gewählt.

Server und Hosting

Für das Hosting wird Microsoft Azure als Cloud-Plattform verwendet. Die Verwaltung der Docker-Container auf diesem Server erfolgt über Docker Compose, was eine effiziente Orchestrierung und Skalierung unserer Anwendungen ermöglicht.

Nginx als Reverse Proxy

Ein wesentlicher Teil der Infrastruktur ist der Einsatz von Nginx als Reverse Proxy. Dieser fungiert als Proxy zwischen den Anfragen der Benutzer (z.B. über das Web) und unserem Server, indem er die Anfragen entgegennimmt und an den entsprechenden laufenden Dienst in Docker-Containern weiterleitet. Dies hat mehrere Vorteile:

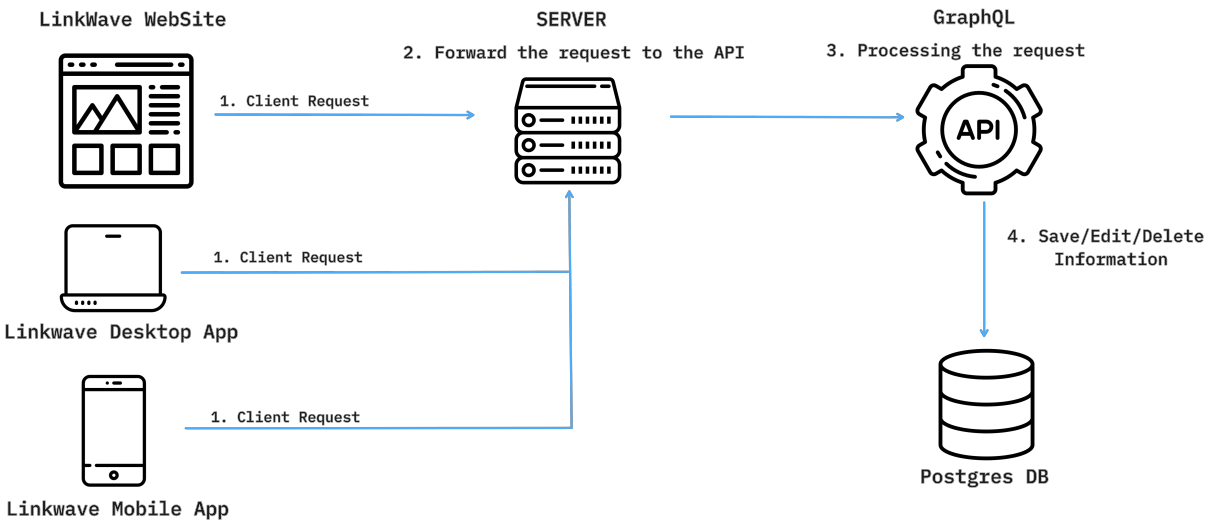


Abbildung 26: Backend Architektur

- **Sicherheit:** Nginx verbirgt die Identität des Backend-Servers, was die Sicherheit erhöht, da Angreifer keine direkten Informationen über den internen Server erhalten.
- **SSL-Verschlüsselung:** Nginx ermöglicht eine einfache Konfiguration von SSL/TLS, um eine sichere Verbindung zwischen Client und Server zu gewährleisten.

Der Nginx-Server von LinkWave ist so konfiguriert, dass alle HTTP-Anfragen auf den sicheren HTTPS-Kanal umgeleitet werden, wie im folgenden Auszug dargestellt:

```
server {
    listen 80;
    server_name api.linkwave.org;
    return 301 https://$host$request_uri;
}
```

SSL-Verschlüsselung

Es werden ausschließlich moderne und sichere SSL-Protokolle und Verschlüsselungen verwendet, um die Datenintegrität zu gewährleisten:

```
server {
```

```
    listen 443 ssl;
    server_name api.linkwave.org;
    ssl_certificate /etc/letsencrypt/live/api.linkwave.org/
       fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/api.linkwave.org/
       privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
       SHA256:...;
}
```

Sicherheitsheader

Um das System gegen gängige Angriffsvektoren zu schützen, wurden die folgenden Sicherheitsheader implementiert:

```
add_header X-Frame-Options "SAMEORIGIN";
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";
```

Proxy-Konfiguration

Nginx fungiert als Proxy, um die Sicherheit zu erhöhen, indem Anfragen an Dienste in laufenden Docker-Containern auf einen intern erreichbaren Port weitergeleitet werden.

```
location / {
    proxy_pass https://localhost:8082;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
```

Sicherheit der Server

Auf dem Server werden zwei Benutzerkonten eingerichtet: ‘linkwave-admin‘ für administrative Aufgaben und ‘linkwave-web‘ für Webanwendungen und Hosting. Benutzer:innen von ‘linkwave-web‘ haben eingeschränkte Rechte, was dem Least-Privilege-Prinzip entspricht und die Sicherheit des Systems erhöht. Durch die Aufteilung der Benutzer:innenrollen wird sichergestellt. dass nur autorisierte Aktionen durchgeführt werden können. Derart wird das Risiko von Sicherheitsverletzungen minimiert.

5 Implementierung der Funktionen

In diesem Abschnitt wird die Implementierung der einzelnen Funktionen von LinkWave auf den verschiedenen Betriebssystemen beschrieben. Dabei wird auf die genaue Funktionsweise und die Einschränkungen auf den verschiedenen Betriebssystemen eingegangen.

5.1 Geräteerkennung

Die Geräteerkennung ermöglicht es Benutzer:innen, andere Geräte in ihrer Umgebung zu finden und mit ihnen zu kommunizieren, ohne diese manuell hinzufügen zu müssen.

Die Geräteerkennung ist in der LinkWave App auf allen Betriebssystemen implementiert. Sie ist für die Benutzung von allen LinkWave Funktionen unentbehrlich. Für die erleichterte Benutzung von LinkWave ist zudem sehr wichtig, dass die Geräteerkennung automatisch, ohne Benutzerinteraktion und zuverlässig abläuft.

Dies ist auch ein Alleinstellungsmerkmal im Vergleich zu anderen “Connectivity-Apps” wie beispielsweise KDE-Connect. In diesen müssen Geräte manuell hinzugefügt werden. Dies ist bei LinkWave nicht nötig, da die Geräteerkennung automatisch abläuft.

5.1.1 Funktionsweise

Die nachfolgende Abbildung veranschaulicht den Einsatz von Bonjour bzw. mDNS durch LinkWave, um die Erkennung und Kommunikation mit anderen Geräten im Netzwerk zu ermöglichen.

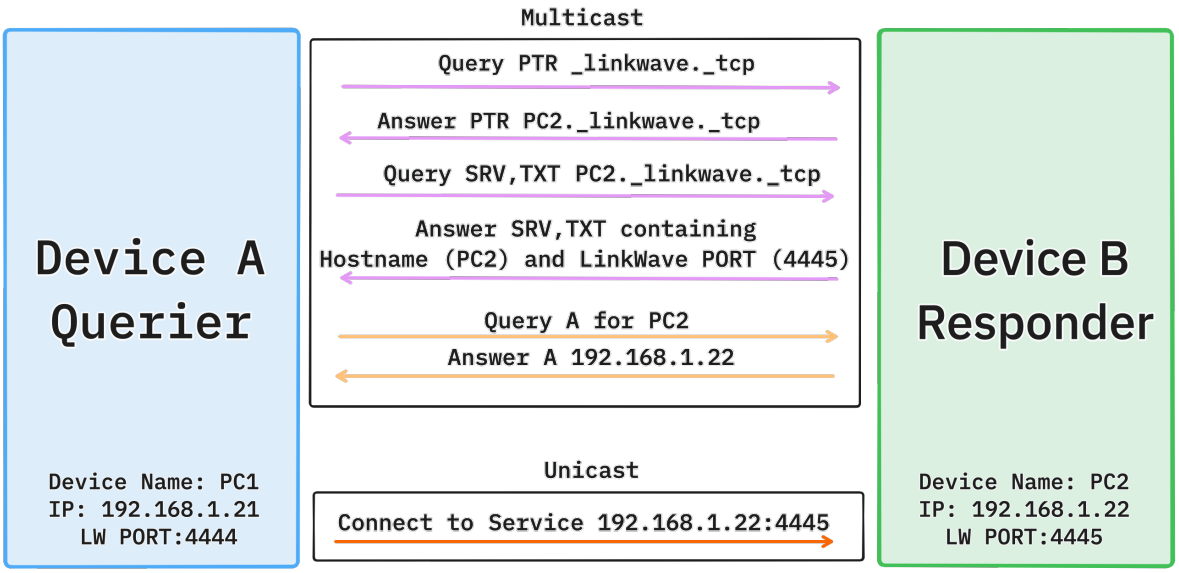


Abbildung 27: Geräteerkennung

1. **mDNS-Anfrage:** Ein Gerät (Device A) sendet eine mDNS-Anfrage im Multicast-Modus, um Dienste zu finden, die "_linkwave._tcp"verwenden. Diese Anfrage ist für alle Geräte im Netzwerk sichtbar.

2. **mDNS-Antwort:** Ein anderes Gerät (Device B) antwortet mit einer PTR (Pointer) Record-Antwort, die den Namen des Dienstes ("PC2._linkwave._tcp") enthält.

3. **Service- und TXT-Records:** Device A fragt dann nach SRV (Service) und TXT (Text) Records, um den Hostnamen (PC2) und den Port des LinkWave-Dienstes (4445) sowie weitere Informationen wie z.B. den Public Key zu erhalten.

4. **A-Record-Anfrage:** Um die IP-Adresse des Dienstes zu finden, sendet Device A eine A-Record-Anfrage für "PC2".

5. **A-Record-Antwort:** Device B antwortet mit der IP-Adresse "192.168.1.22".

6. **Verbindungsaufbau:** Schließlich stellt Device A eine Unicast-Verbindung zum LinkWave-Dienst her, indem es sich mit der IP-Adresse und dem spezifizierten Port ("192.168.1.22:4445") verbindet. Der Port wird dynamisch gewählt, basierend auf den verfügbaren Ports im Betriebssystem, die für den LinkWave TCP Listener verwendet werden.

5.1.2 Unter iOS und macOS

Bonjour bzw. zero-configuration networking ist in iOS und macOS zusammen implementiert und wird über die Klassen `NWBrowser` und `NWListener` von Apple bereitgestellt. Als Urheber von Bonjour stellt Apple Klassen zur leichten Benutzung zur Verfügung. Die Geräteerkennung und das Advertisen des eigenen Gerätes ist in der LinkWave von der Klasse `LinkWaveHelper` implementiert.

Erkennung von Geräten Die Geräteerkennung über Bonjour lässt sich in Swift sehr einfach durch die NWBrowser Klasse benutzen. Im ersten Schritt wird eine Instanz der NWBrowser Klasse erstellt und konfiguriert. Die Konfiguration beinhaltet den Service-Typ, das Protokoll und die Domäne. Wie LinkWave Bonjour im Detail einsetzt ist in Kapitel 5.1.2 beschrieben.

```
let browser: NWBrowser = .init(
  for: .bonjourWithTXTRecord(
    type: "_linkwave._tcp",
    domain: "local"
  ),
  using: .tcp
)
browser.browseResultsChangedHandler = ResultsChangedEventHandler
```

Um die Geräteerkennung nutzen zu können, wird noch ein Event-Handler benötigt, der auf neue bzw. verschwundene Geräte reagiert. Dieser wird über die Funktion `ResultsChangedHandler` definiert. In dieser Funktion erhält man eine Änderungsmenge, die die hinzugefügten und entfernten Geräte enthält. Jede Änderung wird dann in einer Schleife durchgegangen und entsprechend verarbeitet. Die Änderung ist eine Auswahl des `NWBrowser.Result.Change` Enums. In der Schleife kann man aus der Änderung den Endpoint des Gerätes entnehmen und das Gerät entsprechend hinzufügen oder entfernen. Das ist möglich, da Enums in Swift assoziierte Werte für verschiedene Fälle haben können.

```
private func ResultsChangedEventHandler(
  results _: Set<NWBrowser.Result>,
  changes: Set<NWBrowser.Result.Change>
) {
  for change in changes {
    switch change {
      case let .added(endpoint):
      // Geraet zur Liste der sichtbaren Geraete hinzufuegen
      case let .removed(endpoint):
      // Geraet entfernen, da es nicht mehr sichtbar ist
    }
  }
}
```

Aus diesem Endpoint können zusätzliche Informationen wie der Name des Gerätes oder die IP-Adresse entnommen werden. LinkWave verwendet zudem einen TXT-Record, um zusätzliche Informationen wie den Gerätetyp zu übermitteln. TXT-Records werden in dem Endpoint unter `endpoint.metadata.dictionary` gespeichert.

Nach diesen Abfragen wird der Endpoint mit den zusätzlichen Informationen in ein LinkWaveDevice Objekt verpackt und einer Liste hinzugefügt, bzw. aus der Liste entfernt. Diese Liste wird dann an die UI übergeben, um die Geräte anzuzeigen.

Das LinkWaveDevice Objekt enthält die folgenden Informationen:

- **Name:** Eindeutiger Identifikator für jedes Gerät.
- **Type:** Beschreibt die Art des Gerätes (z.B. Windows, Mac, Android).
- **Origin:** Die Herkunft des Gerätes. Dieser Wert ist ein Enum, der die Werte “Network” und “Bluetooth” annehmen kann. Der Wert “Network” bedeutet, dass das Gerät über das Netzwerk erkannt wurde. Der Wert “Bluetooth” ist für zukünftige Erweiterungen gedacht. Im Enum Wert “Network” ist außerdem der Endpoint des Gerätes gespeichert.

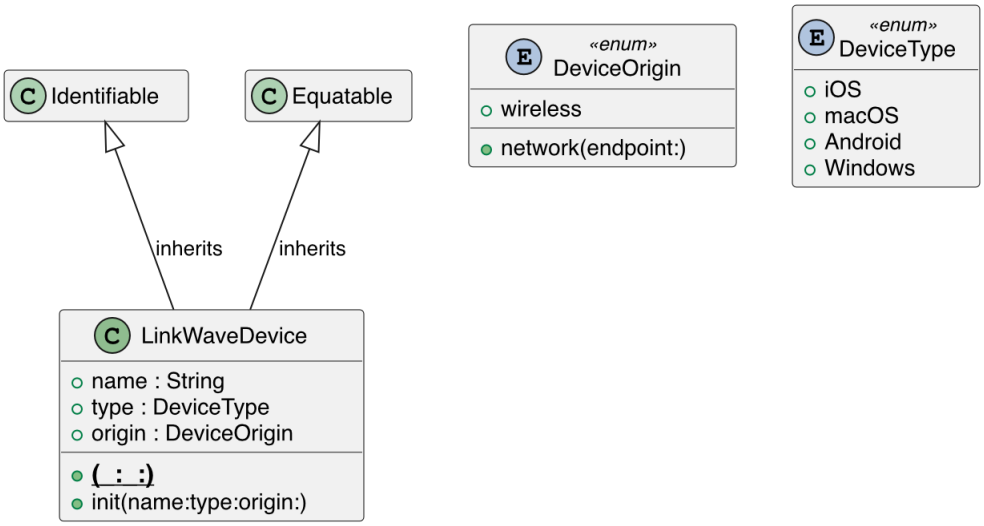


Abbildung 28: UML von LinkWaveDevice

Advertise des eigenen Gerätes Das Advertisen des eigenen Gerätes ist in LinkWave ebenfalls über die NWListener Klasse von Apple implementiert. Der NWListener wird mit einem Service-Typ und einem zufälligen Port initialisiert. Der Service-Typ ist der gleiche wie bei der Geräteerkennung, da alle LinkWave Geräte den gleichen Service-Typ verwenden.

```
let listener: NWListener = try! NWListener(using: .tcp, on: .any)
listener.service = .init(
  name: deviceInfo.deviceName,
  type: "_linkwave._tcp",
  domain: "local"
)
```

Der NWListener wird dann mit einem TXT-Record konfiguriert, der zusätzliche Informationen wie den Gerätetyp enthält. Mit der Compiler-Direktive `#if os(iOS)` wird der Gerätetyp auf iOS oder macOS gesetzt. Basierend auf dem Betriebssystem wird die Variable `osCode` auf “iOS” oder “macOS” gesetzt. Der TXT-Record wird als Dictionary übergeben.

```
#if os(iOS)
  let osCode = "iOS"
#elseif os(macOS)
  let osCode = "macOS"
#endif
let txtRecord = ["deviceType": osCode]
listener.service.txtRecord = .init(txtRecord)
listener.newConnectionHandler = NewConnectionHandler
```

Dem NWListener wird ein NewConnectionHandler übergeben, der auf neue Verbindungen reagiert. In diesem Handler wird die Verbindung angenommen. Dort werden dann die verschiedenen Funktionen von LinkWave aufgerufen. Es ist wichtig, dass dieser Handler auf einem Hintergrund-Thread ausgeführt wird, da er sonst die UI blockieren würde.

```
private func NewConnectionHandler(newConnection: NWConnection) {
  DispatchQueue.global(qos: .background).async {
    newConnection.stateUpdateHandler = { newState in
    switch newState {
      case .ready:
        newConnection.receive() { content, _, _, _ in
          let linkWaveCommandRequest = JSONDecoder()
            .decode(LinkWaveCommandRequest.self, from: content!)
          swift linkWaveCommandRequest.RequestType {
            case .sendFile: // Datei empfangen
            // andere Anfragen
} } } } }
```

Eingehende Verbindungen senden ein JSON-Objekt, das in ein LinkWaveCommandRequest Objekt dekodiert wird. Dieses Objekt enthält Informationen über den Typ des Requests. Basierend auf dem Inhalt des Requests wird dann die entsprechende Funktion aufgerufen.

5.1.3 Unter Android

Bonjour bzw. zero-configuration networking ist in Android als Network Service Discovery Protokoll (NSD) implementiert und wird über die `NsdManager` Klasse bereitgestellt. Die Geräteerkennung und das Advertisen des eigenen Gerätes ist in der LinkWave App in zwei Klassen implementiert: `BonjourController` und `BonjourAdvertiser`.

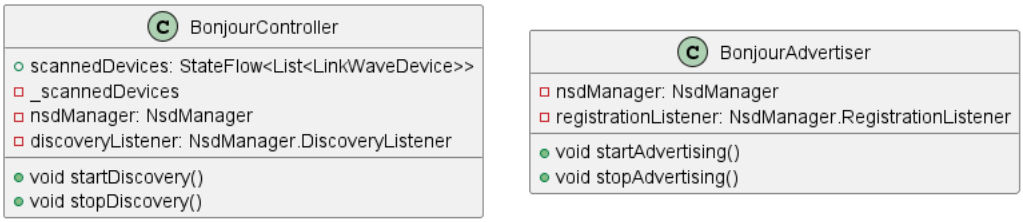


Abbildung 29: UML von BonjourController und BonjourAdvertiser

BonjourController

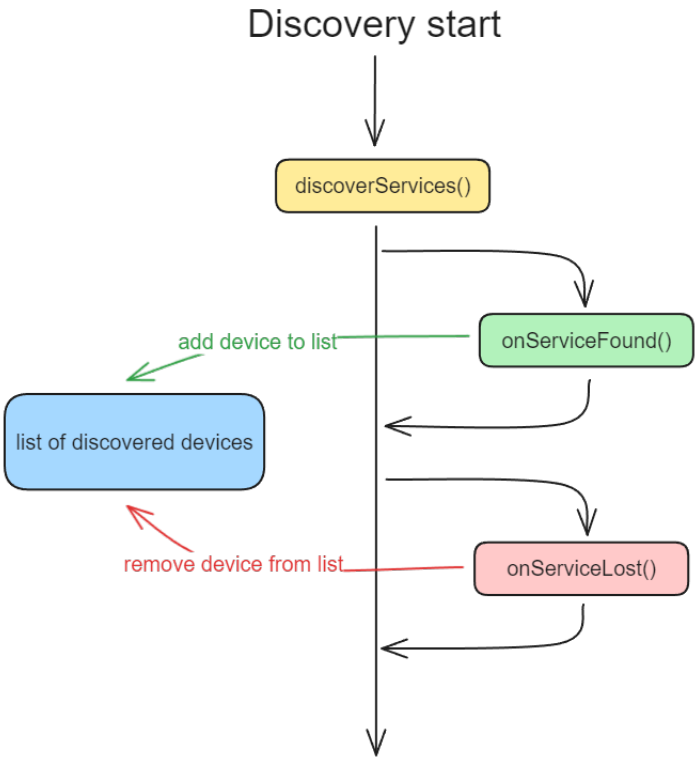


Abbildung 30: Prozess der Geräteerkennung

Der `BonjourController` ist für das Scannen nach Geräten im lokalen Netzwerk zuständig und ist über Dependency Injection in den ViewModels der App verfügbar. In Abbildung 30 ist der Prozess der Geräteerkennung dargestellt. Die über den NsdManager verfügbare Methode `discoverServices()` startet den Discovery-Prozess. Sie bekommt einen `discoveryListener` übergeben, der Callback-Methoden für den Discovery Prozess bereitstellt über die dann die gefundenen Geräte verarbeitet werden können.

Initialisierung Die Instanz des NsdManagers wird über den `androidContext` der App initialisiert. Der `androiContext` enthält Informationen über das "application environment". Der Context wird hier verwendet um auf den System-Service NSD zugreifen zu können. Die folgende Abbildung zeigt den Code zur Initialisierung des NsdManagers.

```
private val nsdManager by lazy {
    androidContext
        .getSystemService(Context.NSD_SERVICE) as NsdManager
}
```

discoverServices() Der BonjourController stellt die Methode `startDiscovery()` bereit, mit der die Geräteerkennung gestartet wird. Diese Methode ruft die `discoverServices()` Methode des `NsdManager` auf, um den Discovery-Prozess zu starten. Hierbei wird der Service-Typ, das Protokoll und der `discoveryListener` übergeben.

```
fun startDiscovery() {
    [...]

    nsdManager.discoverServices(
        BonjourConfig.SERVICE_TYPE,
        NsdManager.PROTOCOL_DNS_SD,
        discoveryListener
    )
}
```

- **SERVICE_TYPE:** Der Service-Typ ist die Service-Identifikation mit dem die Geräte gefunden werden können. Ähnlich wie der hostname in traditionellem DNS, nur wird hier nicht ein einzelner Host identifiziert, sondern alle Hosts, die einen bestimmten Service anbieten. Alle LinkWave Geräte verwenden den gleichen Service-Typ: `_linkwave._tcp`.
- **PROTOCOL_DNS_SD:** Das Protokoll, das für die Geräteerkennung verwendet wird. Hier wird das DNS Service Discovery Protokoll verwendet.
- **discoveryListener:** Die Implementierung des `NsdManager.DiscoveryListener` Interface. Dieses Interface stellt Callback-Methoden für den Discovery-Prozess bereit: Wenn die Suche nach Services gestartet wird, wenn ein Service gefunden wird oder verloren geht und wenn ein Fehler auftritt. Verloren gehen heißt in diesem Kontext, dass ein Service nicht mehr verfügbar ist. Abbildung 31 zeigt die Methoden des `DiscoveryListener`.

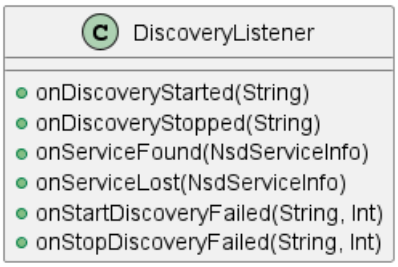


Abbildung 31: UML von DiscoveryListener

onServiceFound() Wenn ein Gerät gefunden wurde wird die `onServiceFound()` Methode des `discoveryListeners` aufgerufen. Jetzt müssen die Connection Informationen, wie die IP-Adresse und die Portnummer mittels der `nsdManager.resolveService()` Methode abgerufen werden. Auch diese Methode bekommt wieder ein Objekt mit Callback Methoden übergeben, die aufgerufen werden, wenn die Informationen abgerufen wurden oder ein Fehler auftritt. Hierfür kann seit Android 14 eine Implementierung des `NsdManager.ServiceInfoCallback` Interface verwendet werden. Anders als der veraltete `NsdManager.ResolveListener` kann der `ServiceInfoCallback` darauf reagieren,

wenn sich ein Attribut eines Services ändert. Wenn sich also die IP-Adresse von einem Gerät ändert, kann man über den ServiceInfoCallback darauf reagieren. Da dieses Interface jedoch erst ab Android 14 verfügbar ist, verwendet LinkWave noch den `ResolveListener`. Der `ResolveListener` bekommt ein `onServiceResolved()` Callback, der aufgerufen wird, wenn ein bestimmter Service aufgelöst wurde. Sollte sich ein Attribut eines Gerätes ändern, müsste es neu aufgelöst werden, da der `ResolveListener` nach dem ersten Auflösen verworfen wird und somit nicht auf Änderungen des aufgelösten Services reagieren kann.

```
override fun onServiceFound(serviceInfo: NsdServiceInfo?) {
    [...]
    val resolveListener = ResolveListener { resolvedServiceInfo ->
        _scannedDevices.update { devices ->
            val newDevice = BonjourDevice
                .fromNsdServiceInfo(resolvedServiceInfo)
            if (newDevice in devices) devices
            else devices + newDevice
        }
    }

    nsdManager.resolveService(serviceInfo, resolveListener)
}
```

onServiceLost() Wenn ein Gerät verloren geht, wird die `onServiceLost()` Methode des `discoveryListeners` aufgerufen. In dieser Methode wird das Gerät aus der Liste der gefundenen Geräte entfernt.

```
override fun onServiceLost(serviceInfo: NsdServiceInfo?) {
    [...]
    _scannedDevices.update { devices ->
        devices.filter { it.name != serviceInfo.serviceName
            && it.address != serviceInfo.host }
    }
}
```

aktualisieren der Benutzeroberfläche Die Liste der gefundenen Geräte wird über einen `StateFlow` in den ViewModels der App bereitgestellt. Der `BonjourController` aktualisiert diesen `StateFlow` mit den gefundenen Geräten. Die ViewModels der App können sich auf diesen `StateFlow` subscriben und so immer die aktuelle Liste der gefundenen Geräte erhalten. Kotlin Flow ist eine Bibliothek, die es ermöglicht, asynchrone Datenströme zu verarbeiten. Der folgende Code zeigt, wie die ViewModels in den Activities der App verwendet werden, um die gefundenen Geräte anzuzeigen.

```
class ShareActivity() : ComponentActivity() {
    private val devicesViewModel: DevicesViewModel by viewModel()

    setContent {
        val devicesState = bonjourViewModel.state
            .collectAsState()

        DeviceList(
            devices = bonjourState.value.scannedDevices,
        )
    }
}
```

BonjourAdvertiser

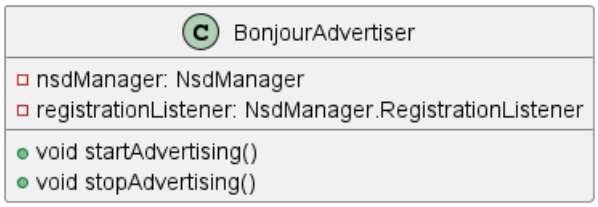


Abbildung 32: Prozess der Geräteerkennung

Der `BonjourAdvertiser` ist für das Bekanntmachen des Gerätes im lokalen Netzwerk zuständig. Er wird vom `LinkWaveService`, der in Kapitel 5.5.3 genauer beschrieben wird, verwaltet: Wenn der Service startet wird der `BonjourAdvertiser` initialisiert und das eigene Gerät wird im lokalen Netzwerk bekannt gemacht.

startAdvertising() Der `BonjourAdvertiser` wird beim starten des `LinkWaveService` initialisiert. Hierfür wird die `startAdvertising()` Methode aufgerufen, die mittels des `NsdManager` einen Service mit dem Namen des Gerätes registriert. Die Informationen des Gerätes werden in einem `NsdServiceInfo` Objekt gespeichert und dem `NsdManager` übergeben.

```
val serviceInfo = NsdServiceInfo().apply {
    serviceName = BonjourConfig.serviceName
    serviceType = BonjourConfig.SERVICE_TYPE
    port = BonjourConfig.SERVICE_PORT
    setAttribute("deviceType", LocalDeviceInfo.type.name)
}
```

- **serviceName:** Der Name des Services, der im lokalen Netzwerk angezeigt wird. Hier wird der Name des Gerätes verwendet.
- **serviceType:** Der Service-Typ, der im lokalen Netzwerk angezeigt wird. Hier wird der Service-Typ von LinkWave verwendet: `_linkwave._tcp`.
- **port:** Der Port, auf dem der Service erreichbar ist. LinkWave verwendet einen random Port.

- **setAttribute:** Hier können zusätzliche Informationen über das Gerät gespeichert werden. Hier wird der Gerätetyp des Gerätes gespeichert.

Der Prozess um das Gerät im Netzwerk bekannt zu geben wird mit der `registerService` Methode des `NsdManager` gestartet. Dieser Methode wird das `NsdServiceInfo` Objekt und ein `registrationListener` übergeben. Der `registrationListener` stellt Callback Methoden zur Verfügung, die aufgerufen werden, wenn der Prozess erfolgreich war oder ein Fehler aufgetreten ist.

Abbildung 33: UML von RegistrationListener

```
suspend fun startAdvertising() = withContext(Dispatchers.IO) {
    val serviceInfo = [...]

    nsdManager.registerService(
        serviceInfo,
        NsdManager.PROTOCOL_DNS_SD,
        registrationListener
    )
}
```

stopAdvertising() Der `BonjourAdvertiser` stellt die Methode `stopAdvertising()` bereit, um das eigene Gerät wieder aus dem lokalen Netzwerk zu entfernen. Hierfür wird die `unregisterService()` Methode des `NsdManager` aufgerufen.

```
suspend fun stopAdvertising() = withContext(Dispatchers.IO) {
    nsdManager.unregisterService(registrationListener)
}
```

5.1.4 Unter Windows

Für die Geräteerkennung unter Windows wird ein Library namens “Zeroconf” verwendet. Die “Zeroconf” Library ermöglicht es, mDNS-Anfragen zu senden und mDNS-Antworten zu empfangen. Die Geräteerkennung wird durch die Klassen `Discovery` und `Advertise` implementiert. Die Methoden dieser Klassen werden in der Klasse `LinkWaveServie` in LinkWaveUI.Core verwendet bzw. aufgerufen. Die Klasse `Discovery` ist für das Scannen nach Geräten im lokalen Netzwerk zuständig und die Klasse `Advertise` ist für das Advertisen des eigenen Gerätes zuständig.

Discovery

Die Funktion FindLinkWaveDevices() verwendet die ZeroconfResolver Bibliothek, um Geräte zu finden, die LinkWave-Dienste im lokalen Netzwerk anbieten (andere Geräte, die LinkWave installiert haben). Dazu wird ein Listener erstellt, der nach mDNS (Multicast DNS) Einträgen sucht, die unter dem Servicenamen “_linkwave._tcp.local” veröffentlicht sind.

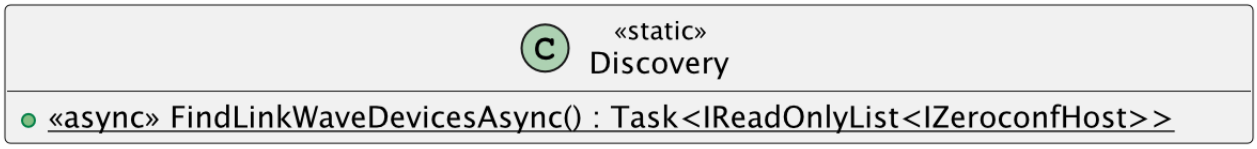


Abbildung 34: UML Discovery

```
public static class Discovery
{
    public static ZeroconfResolver.ResolverListener
        FindLinkWaveDevices()
    {
        var options = ZeroconfResolver
        .CreateListener("_linkwave._tcp.local.");
        return options;
    }
}
```

Advertise

Die Methode RegisterService-Methode() verwendet den Service Namen “_linkwave._tcp”, um eine Dienstbeschreibung zu erstellen, die dann im Netzwerk bekannt gemacht wird. Die ServiceProfile Instanz wird mit dem gegebenen Instanznamen, dem Service Namen und dem Port konfiguriert. Anschließend wird diese Konfiguration mithilfe der Advertise-Methode im lokalen Netzwerk angekündigt, so dass andere Geräte den Dienst entdecken und mit ihm interagieren können.

Abbildung 35: UML Advertisement

```
public class Advertisement
{
    public static async Task RegisterService(string instance,
       ushort port)
    {
        string serviceName = "_linkwave._tcp";
        var service = new ServiceProfile(instance, serviceName,
           port);
        // Service Felder setzen
        sd.Advertise(service);
    }
}
```

5.1.5 Am Server

Im folgenden werden die GraphQL Endpoints für die Geräteerkennung auf dem Server beschrieben:

Datenbankstruktur

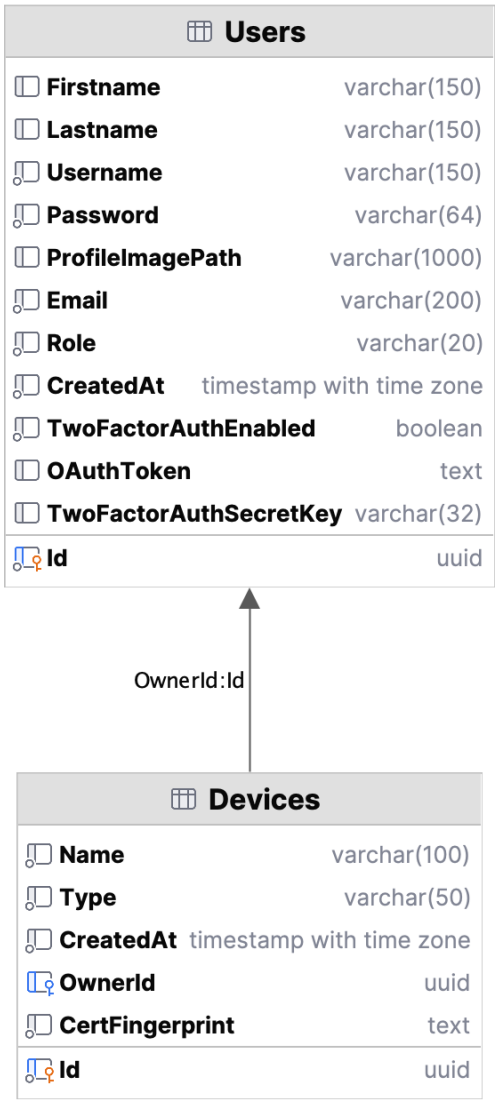


Abbildung 36: Devices Tabelle

Die Tabelle besteht aus mehreren Feldern, die zusammenarbeiten, um umfassende Daten zu jedem Gerät zu speichern:

- **Id (UUID):** Eindeutiger Identifikator für jedes Gerät.
- **Name (String):** Name des Gerätes, bereitgestellt vom Benutzer zur einfachen Erkennung.
- **Type (String):** Beschreibt die Art des Gerätes (z.B. Windows, Mac, Android).
- **Owner (User):** Verweist auf den Eigentümer des Gerätes in der 'Users' Tabelle.
- **CertFingerprint (String):** Ein Textfeld, das den Fingerprint des Zertifikats speichert, das bei der Registrierung eines Kontos auf einem Gerät generiert wird.
- **CreatedAt (DateTime):** Das Datum und die Uhrzeit der Erstellung des Geräteeintrags.

GraphQL-Endpunkte für Devices

AddDevice Über den GraphQL-Mutations-Endpunkt `AddDevice` können Benutzer neue Geräte zum System hinzufügen. Der Endpunkt erwartet die Geräteinformationen und speichert sie in der 'Devices' Tabelle. Der zugehörige Benutzer wird aktualisiert, um das neue Gerät in seiner Geräteliste zu reflektieren.

```
public async Task<Database.Models.Device> AddDevice(LinkWaveContext
    dbContext, Guid userId , string deviceName, string deviceType,
   string fingerprint)
{
    ...
    dbContext.Devices.Add(device);
    user.Devices.Add(device);
    return device;
    // Error Handling wurde ausgelassen
}
```

GetDevicesByUserId Der Query Endpoint `GetDevicesByUserId` ermöglicht es Benutzer:innen, eine Liste aller registrierten Geräte abzurufen, die ihrem Konto zugeordnet sind. Dieser Endpunkt liefert eine Sammlung von Gerätedaten, die für die Verwaltung oder Überprüfung durch den Benutzer:innen verwendet werden kann.

```
public async Task<List<Database.Models.Device>> GetDevicesByUserId(
    LinkWaveContext dbContext, Guid userId)
{
     var usersDevices = await dbContext.Users.SelectMany(x => x.
         Devices).Where(x => x.Owner.Id == userId).ToListAsync();
     return usersDevices;
}
```

UpdateDeviceDetails Dieser Endpoint `UpdateDeviceDetails` dient dazu, die Daten eines bestehenden Geräts zu aktualisieren. Dies kann für verschiedene Szenarien erforderlich sein, wie beispielsweise das Aktualisieren des Zertifikatsfingerabdrucks oder des Gerätenamens, um die Gerätedaten auf dem neuesten Stand zu halten und die Systemintegrität zu sichern.

```
public async Task<Database.Models.Device> UpdateDeviceDetails(
    LinkWaveContext dbContext, Guid deviceId, string newDeviceName,
    string newCertFingerprint)
{
     ...
     deviceToUpdate.CertFingerprint = newCertFingerprint;
     dbContext.Devices.Update(deviceToUpdate);
     return deviceToUpdate;
}
```

DeleteDevice Der `DeleteDevice`-Endpunkt ist für die Entfernung von Geräten aus dem System verantwortlich.

```
public async Task<bool> DeleteDevice(LinkWaveContext dbContext,
    Guid deviceId)
{
     ...
     dbContext.Devices.Remove(deviceToRemove);
     await dbContext.SaveChangesAsync();
     return true;
}
```

5.2 Dateiübertragung

Die Dateiübertragung ist eine der wichtigsten Funktionen von LinkWave. Sie ermöglicht es, Dateien zwischen verschiedenen Geräten zu übertragen. Die Dateiübertragung funktioniert auf allen Betriebssystemen ohne Einschränkungen.

Um die Erfahrung für die Benutzer:innen so angenehm wie möglich zu machen wurde ein einheitlicher Ablauf gestaltet. Der empfohlene Ablauf der Dateiübertragung ist wie folgt:

1. **Teilen:** Beim Benutzen von anderen Apps muss man bei Dateien, die man übertragen möchte, die systemspezifische Teilen-Funktion benutzen.
2. **LinkWave auswählen:** In dem sich öffnenden Menü kann man LinkWave auswählen. Sobald LikWave installiert ist, fügt es sich automatisch zu diesem Menü hinzu.
3. **Gerät auswählen:** Nun muss man das Gerät, an das die Dateien gesendet werden sollen, auswählen. Dieses Gerät muss sich in der Nähe befinden, da die Dateiübertragung über eine direkte Verbindung zwischen den Geräten erfolgt.
4. **Bestätigen:** Sobald das Gerät ausgewählt wurde bekommt man auf dem Gerät, an das die Dateien gesendet werden sollen, eine Benachrichtigung. In dieser Benachrichtigung kann man die Dateiübertragung bestätigen oder ablehnen. Sobald bestätigt wurde, wird die Dateiübertragung gestartet. Die Datei erscheint nach der Übertragung sofort auf dem in den Einstellungen ausgewählten Speicherort.

Mit diesen vier Schritten wird derselbe Arbeitsablauf wie bei Apple “Airdrop” ermöglicht. Dieses überzeugt stark durch seine Simplizität und ist deswegen ein Maßstab für LinkWave.

5.2.1 Funktionsweise

Die File-Sharing-Funktion von LinkWave ist als sicherer und benutzerfreundlicher Prozess konzipiert, der auf einem automatisierten und verschlüsselten Kommunikationsablauf basiert. Im Folgenden wird der Funktionsmechanismus von LinkWave für die Freigabe von Dateien beschrieben, wie er in der Applikationsflussdiagramm dargestellt ist:

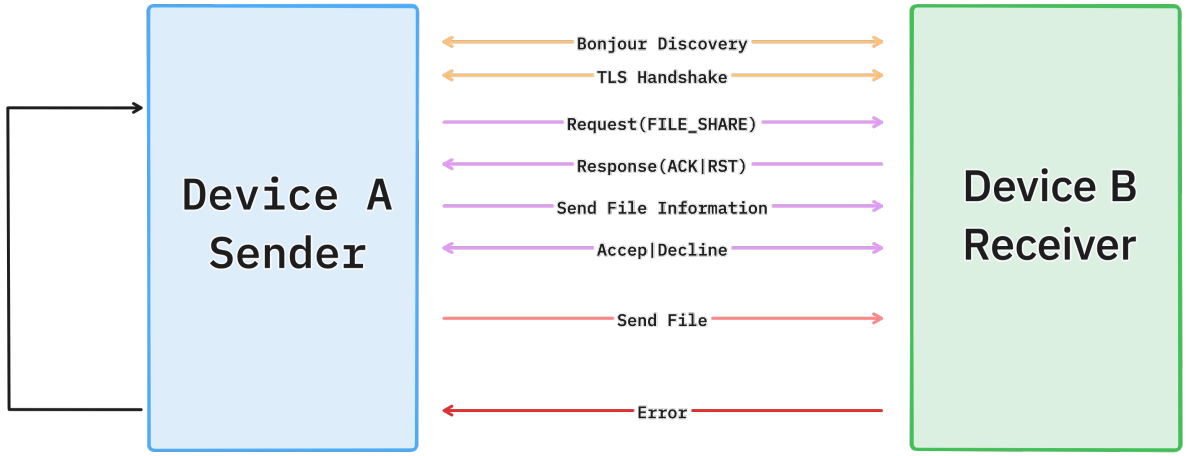


Abbildung 37: File Sharing

1. **Bonjour Discovery:** Der Prozess beginnt mit der Erkennung von Geräten, auf denen LinkWave installiert ist, über das Bonjour-Protokoll. LinkWave verwendet das Bonjour-Protokoll, um Geräte im lokalen Netzwerk automatisch zu erkennen.

2. **TLS Handshake:** Nach der Geräteerkennung und der Auswahl des Empfängers stellt LinkWave eine sichere Verbindung über TLS (Transport Layer Security) her, was durch den TLS Handshake dargestellt wird. Dies gewährleistet eine verschlüsselte Kommunikation zwischen den Geräten und schützt die übertragenen Dateien vor potenziellen Sicherheitsrisiken.

3. **Request (FILE_SHARE):** Das Sendegerät (Device A) signalisiert seine Bereitschaft zum Datenaustausch durch eine entsprechende Anfrage an das Empfangsgerät (Device B), womit der Datenaustausch gestartet wird.

4. **Response (ACK|RST):** Device B antwortet auf die Anfrage entweder mit einer Bestätigung (ACK für Acknowledge) oder mit einem Reset (RST für Reset). Ein ACK setzt den Prozess fort, während ein RST die Anforderung zurückweist und den Prozess beendet.

5. **Send File Information:** Nach erfolgreicher Bestätigung sendet Device A Informationen über die freizugebende Datei an Device B. LinkWave verwendet diese Informationen, um den Empfänger mit Details wie Dateigröße, Typ und Name zu versorgen.

6. **Accept/Decline:** Basierend auf den erhaltenen Dateiinformationen hat Device B die Möglichkeit, die Datei zu akzeptieren oder abzulehnen. LinkWave stellt hierfür eine Benutzeroberfläche zur Verfügung, über die der Empfänger seine Entscheidung treffen kann.

7. **Send File:** Wenn der Empfänger die Datei akzeptiert, beginnt Device A mit der Übertragung der Datei zu Device B. LinkWave verwaltet den Übertragungsprozess und zeigt den Fortschritt der Übertragung an.

8. **Error:** Sollten Fehler während der Übertragung auftreten, stellt LinkWave Mechanismen zur Fehlererkennung und -meldung bereit.

5.2.2 Unter iOS und macOS

Die Dateiübertragung ist in iOS und macOS über TCP implementiert. Die Dateiübertragung wird über die `NWConnection` Klasse von Apple realisiert. Der Code ist auf beiden Betriebssystemen gleich, da die `NWConnection` Klasse in Swift auf beiden Betriebssystemen verfügbar ist.

Datei senden

Um Dateien zu senden kann man die Funktion `sendFileHandler` verwenden. Diese Funktion bekommt folgende Parameter übergeben:

- **Endpoint(NWEndpoint):** Der Endpoint des Gerätes, an das die Datei gesendet werden soll.
- **FileStream(InputStream):** Der Stream der Datei, die gesendet werden soll.
- **FileName(String):** Der Name der Datei, die gesendet werden soll.
- **FileSize(Int64):** Die Größe der Datei, die gesendet werden soll.
- **FileSendProgress(inout Double):** Ein inout Parameter, über den der Fortschritt in % an das UI weitergeleitet wird.
- **AcceptedCallback((Bool)** -> **Void):** Eine Callback-Funktion, die dann aufgerufen wird, wenn die Dateiübertragung akzeptiert oder abgelehnt wurde.
- **CompletionCallback((Bool)** -> **Void):** Eine Callback-Funktion, die dann aufgerufen wird, sobald die Dateiübertragung abgeschlossen wurde.

In dieser Funktion wird die Dateiübertragung nach dem Schema in Kapitel 5.2.1 durchgeführt. Der Stream der Datei wird in kleine Pakete aufgeteilt und nacheinander an das Gerät gesendet. Der Fortschritt der Dateiübertragung wird an das UI weitergeleitet, damit der Benutzer den Fortschritt sehen kann. Wenn die Dateiübertragung abgeschlossen ist, wird die `CompletionCallback` Funktion aufgerufen.

Die von Apple bereitgestellten Funktionen `send()` und `receive()` der `NWConnection` Klasse sind asynchron und blockieren den Haupt-Thread nicht. Mit ihnen werden jegliche Datenpakete übertragen. Um eine NWConnection von einem Bonjour Endpoint zu erstellen, muss man lediglich den Endpoint in den Konstruktor übergeben. Die NWConnection lässt sich dann mit dem `start(queue: .main)` Methodenaufruf auf dem Main-Thread starten. Die Datenpakete werden asynchron übertragen. Die `send()` und `receive()` Methoden funktionieren mit Callbacks, die aufgerufen werden, wenn Daten empfangen oder gesendet wurden.

```
connection.receive(
    minimumIncompleteLength: 1, maximumLength: 1024
) { content, _, _, error in
  // Daten verarbeiten
}
```

Die Datei wird über einen `InputStream` in chunks gelesen und über die `send()` Methode an das Gerät gesendet. Der `InputStream` wird in chunks von 1024 Bytes gelesen und an das Gerät gesendet. Der Fortschritt der Dateiübertragung wird an das UI weitergeleitet, damit der Benutzer den Fortschritt sehen kann. Dies wird in einer Schleife durchgeführt, bis die gesamte Datei übertragen wurde. Dank der Callback Funktionsweise muss man nicht warten, bis die Übertragung eines Chunks abgeschlossen ist, sondern kann sofort den nächsten senden um die Übertragung zu beschleunigen.

```
var buffer = [UInt8](repeating: 0, count: transferChunkSize)
while case let amount = fileStream.read(
  &buffer,
  maxLength: transferChunkSize),
  amount > 0 {
  let chunk = Array(buffer[..<amount])
  connection.send(content: chunk) { error in
    // Bei erfolgreicher Uebertragung UI aktualisieren
  }
}
```

Datei empfangen

Das Empfangen von Dateien ist in der LinkWave App von der Klasse `LinkWaveHelper` implementiert. Dies ist eine weitere Funktion, die im `NewConnectionHandler` (siehe Kapitel 5.1.2) aufgerufen werden kann.

Nach dem Aufbau der Verbindung werden die Informationen des zu empfangenden Files empfangen. Diese Informationen enthalten den Dateinamen und die Dateigröße. Anschließend wird ein Popup angezeigt, in dem der Benutzer die Dateiübertragung akzeptieren oder ablehnen kann. Zudem wird ein Callback registriert, der aufgerufen wird, wenn die Dateiübertragung angenommen/abgelehnt wird. Wenn der Benutzer die Dateiübertragung akzeptiert, wird die Datei empfangen und gespeichert. Der Fortschritt der Dateiübertragung wird an das UI weitergeleitet, damit der Benutzer den Fortschritt sehen kann.

5.2.3 Unter Android

In folgendem Abschnitt wird die Implementierung der Dateiübertragung in der Android App beschrieben. Die Dateiübertragung wird über eine TCP-Verbindung realisiert.

Datei senden

Abbildung 38 zeigt den Prozess der Dateiübertragung auf Android.

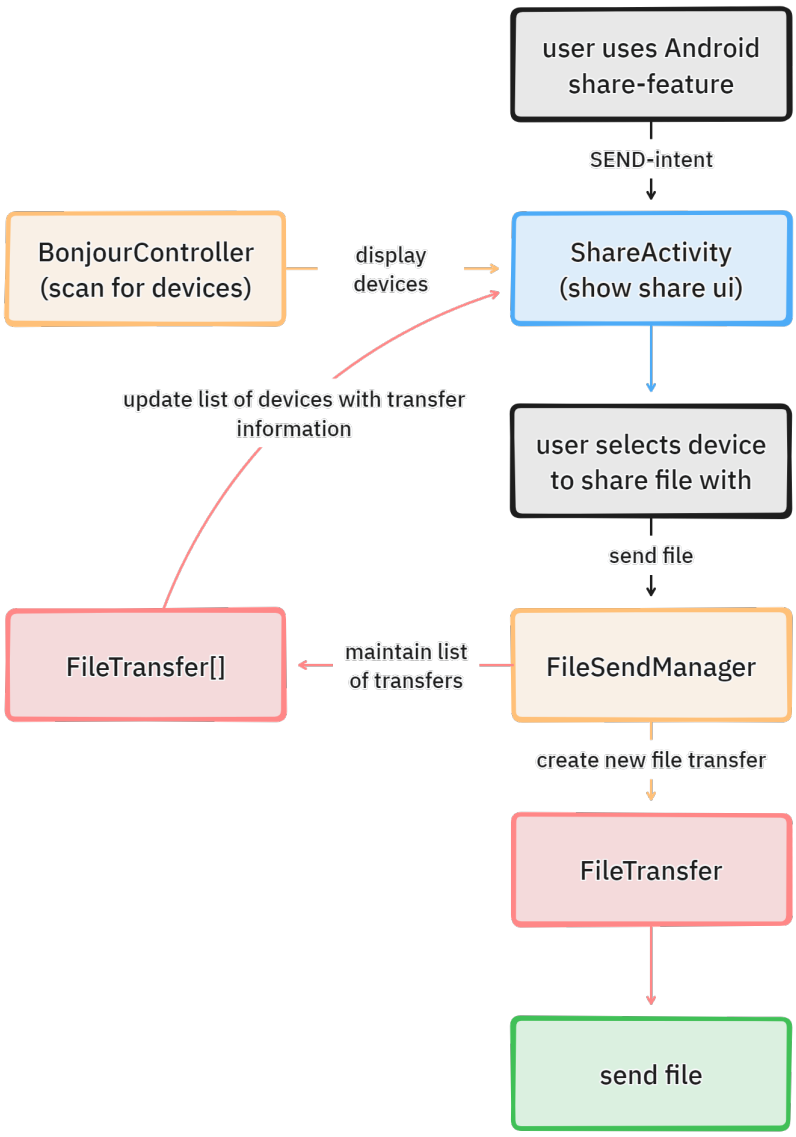


Abbildung 38: Prozess der Dateiübertragung auf Android

Um eine Datei zu teilen, müssen die Benutzer:innen das “Teilen”-Feature von Android verwenden. Wird im “Teilen”-Fenster die LinkWave App ausgewählt, schickt Android einen `SEND`-Intent mit einer URI zu der Datei an die App. Damit die App einen `SEND`-Intent empfangen kann, muss ein `intent-filter` konfiguriert werden. Dieser wird in der `AndroidManifest.xml` Datei definiert.

```
<activity
    android:name=".ui.share.ShareActivity"
    android:excludeFromRecents="true"
    android:exported="true"
    android:launchMode="singleInstance"
    android:theme="@style/Theme.App">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="*/*" />
    </intent-filter>
</activity>
```

Hier ist der `intent-filter` zu sehen, der den `SEND`-Intent abfängt. Der `intent-filter` ist so konfiguriert, dass er alle Dateitypen empfangen kann. Der `mimeType` spezifiziert den Typ der Datei. Der `mimeType` von einem `jpeg` Bild ist `image/jpeg`. `*/*` bedeutet, dass der Filter für alle Dateitypen gilt.

ShareActivity Der `intent-filter` gehört zu der `ShareActivity`. Bekommt die App einen `SEND`-Intent, wird entweder die `onCreate()` Methode oder die `onNewIntent()` Methode aufgerufen. Welche Methode aufgerufen wird, hängt vom `launchMode` und ob sich die Activity bereits im Hintergrund befindet ab. `singleInstance` bedeutet, dass es nur eine Instanz der Activity geben kann. Wenn die Activity bereits im Hintergrund ist, wird die `onNewIntent()` Methode aufgerufen, ansonsten wird die `onCreate()` Methode aufgerufen.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    intent?.let { handleIntent(it) }

    // ... UI initialisieren
}

override fun onNewIntent(intent: Intent?) {
    super.onNewIntent(intent)
    intent?.let { handleIntent(it) }
}
```

handleIntent() In der `handleIntent()` Methode wird der Intent verarbeitet. Die in dem Intent enthaltene URI wird an das `ShareViewModel` übergeben, welches die Daten der Datei ausliest und diese als `selectedFile` zwischen speichert.

```
private fun handleIntent(intent: Intent) {
    when (intent.action) {
        Intent.ACTION_SEND -> {
            val uri = intent.getParcelableExtra(Intent.EXTRA_STREAM
                , Uri::class.java)

            if (uri == null) return

            shareViewModel.selectFile(uri, this)
        }
        [...]
    }
}
```

Die ausgewählte Datei wird in der `ShareActivity` angezeigt. Jetzt muss ein Gerät aus der `DeviceList` ausgewählt werden, an das die Datei gesendet werden soll. Folgender Code-Ausschnitt zeigt, wie die Geräte und die ausgewählte Datei in der `ShareActivity` angezeigt werden.

```
override fun onCreate(savedInstanceState: Bundle?) {
    setContent {
        val fileState = shareViewModel.state.collectAsState()
        val devicesState = devicesViewModel.state.collectAsState()

        FileArea(file = fileState.value.selectedFile)

        DeviceList(
            devices = devicesState.value.scannedDevices,
            fileTransfers = fileState.value.fileTransfers,
            onSelect = { device ->
                shareViewModel.sendFile(device)
            }
        )
    }
}
```

FileSendManager Der `FileSendManager` verwaltet ausgehende Dateiübertragungen. Er wird vom `ShareViewModel` verwendet, um Dateien zu senden. Abbildung 39 zeigt das Klassendiagramm des `FileSendManager`.

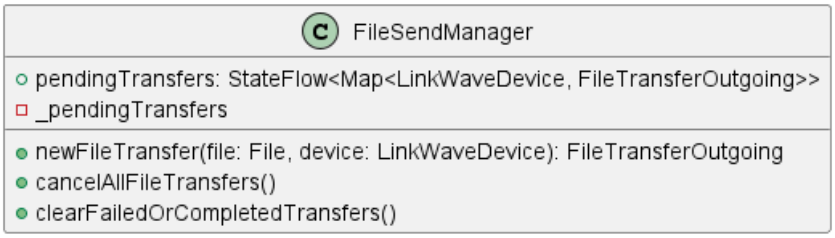


Abbildung 39: UML von FileSendManager

Der `FileSendManager` speichert die aktuelle Dateiübertragung in einer HashMap. Mit der `newFileTransfer()` Methode wird eine neue Dateiübertragung erstellt und in der HashMap gespeichert. Die `send()` Methode des `FileTransfer` wird aufgerufen, um die Dateiübertragung zu starten.

```
fun newFileTransfer(file: AndroidFile, device: LinkWaveDevice) {
    if (_pendingTransfers.value.containsKey(device)) {
        return
    }

    val fileTransfer = FileTransferOutgoing(androidContext, file,
       device)

    _pendingTransfers.update { transfers ->
        transfers + (device to fileTransfer)
    }

    fileSendManagerScope.launch {
        fileTransfer.send()
    }
}
```

FileTransferOutgoing Ein `FileTransferOutgoing` Objekt ist für das Senden einer Datei an ein anderes Gerät zuständig. Es verwendet eine `TCP-Socket` Verbindung, um die Datei zu senden und stellt Informationen über den Status und den Fortschritt der Dateiübertragung bereit.

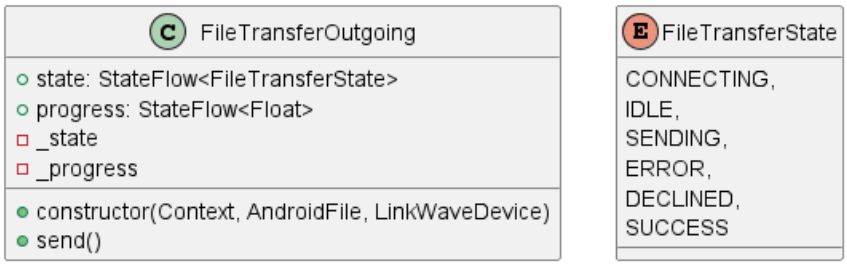


Abbildung 40: UML von FileTransferOutgoing

send() Der genaue Ablauf der Dateiübertragung wird in 5.2.1 beschrieben. Die `Request` Objekte sowie die Dateiinformationen werden als `Json` serialisiert und über die `Socket`

Verbindung an das Empfängergerät gesendet. In folgendem Code-Ausschnitt wird das Senden der Dateiinformationen gezeigt.

```
val outputStream = socket.getOutputStream()

val fileInfo = FileShareRequest(
    file.name,
    file.size,
    file.calculateChecksum(androidContext)
)

outputStream.writeJson(fileInfo)
```

Die `writeJson` Extension-Funktion serialisiert ein beliebiges Objekt als `Json` und schreibt die serialisierten Daten in den `OutputStream`. Hierfür wird die `kotlinx.serialization` Bibliothek [7] verwendet. Um die Antwort auf die `FileShareRequest`-Anfrage zu erhalten wird die `readJson` Extension-Funktion verwendet.

```
val fileShareResponse =
    inputStream.readJson<FileShareResponse>(1024)
```

Wird die Anfrage akzeptiert, wird die Datei in Chunks gelesen und über die `Socket` Verbindung an das Empfängergerät gesendet. Der Fortschritt der Dateiübertragung wird an das UI weitergeleitet, damit der Benutzer den Fortschritt sehen kann. Folgender Code-Ausschnitt zeigt das Senden der Datei.

```
while (stream.read(buffer, 0, buffer.size)
    .also { read -> bytesRead = read } != -1
) {
    outputStream.write(buffer, 0, bytesRead)
    bytesTotalRead += bytesRead

    _progress.value = bytesTotalRead.toFloat() / file.size
}
```

Datei empfangen

Da das Empfangen von Dateien auch im Hintergrund, wenn die App geschlossen ist, funktionieren soll, läuft der Empfangsprozess in dem `LinkWaveService` (siehe 5.5.3). Der Empfangsprozess wird in Abbildung 41 dargestellt.

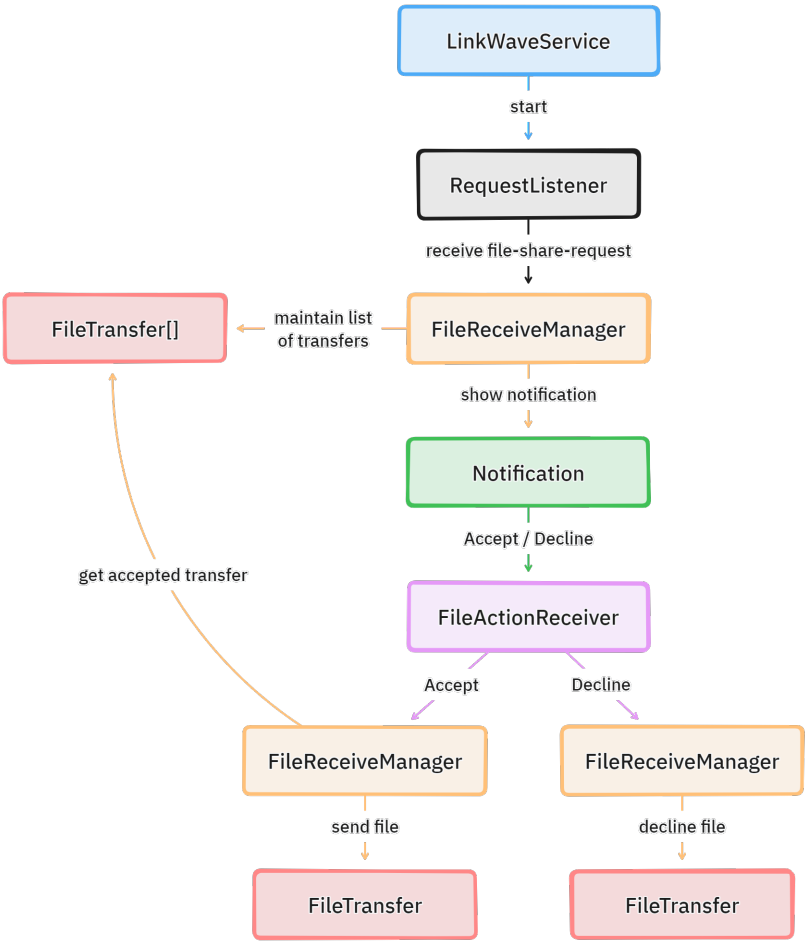


Abbildung 41: Prozess der Dateiübertragung auf Android

RequestListener Der `RequestListener` wird vom `LinkWaveService` in einer eigenen Coroutine gestartet. Er ist dafür verantwortlich, eingehende Anfragen zu verarbeiten. Abbildung 42 zeigt das Klassendiagramm des `RequestListener`.

Abbildung 42: UML von RequestListener

FileReceiveManager Wird eine `FILE_SHARE` Anfrage empfangen, wird diese an den `FileReceiveManager` weitergeleitet. Der `FileReceiveManager` speichert die aktuellen Dateiübertragungen in einer HashMap und zeigt dem Benutzer eine Benachrichtigung an, mit der die Dateiübertragung akzeptiert oder abgelehnt werden kann.

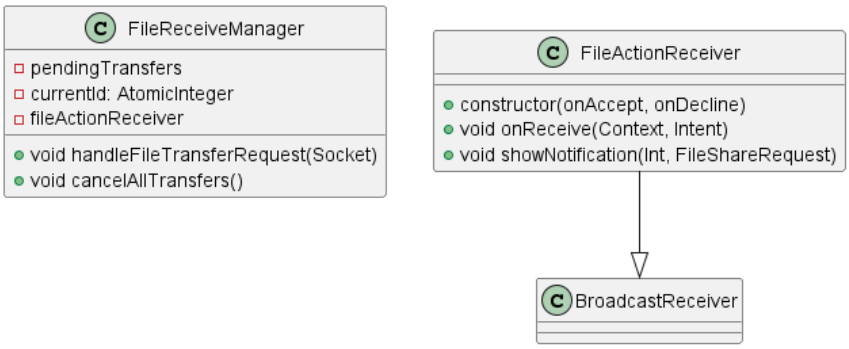


Abbildung 43: UML von FileReceiveManager und FileActionReceiver

FileActionReceiver Der `FileActionReceiver` ist ein `BroadcastReceiver`, der auf das Akzeptieren oder Ablehnen einer Dateiübertragung reagiert. In der Benachrichtigung, die dem Benutzer gezeigt wird, werden die zwei Aktionen festgelegt, die dann beim Auswählen den jeweiligen Intent um die Datei anzunehmen oder abzulehnen an den `FileActionReceiver` senden.

```
override fun onReceive(context: Context?, intent: Intent?) {
    // read id from intent...
    when (intent.action) {
        ACTION_FILE_ACCEPT -> {
            onAccepted(id)
        }

        ACTION_FILE_DECLINE -> {
            notificationManager.cancel(id)
            onDenied(id)
        }
    }
}
```

Die `onAccepted()` und `onDenied()` Callback-Methoden werden im `FileReceiveManager` festgelegt. Die `onAccepted()` Funktion wird aufgerufen, wenn der Benutzer die Dateiübertragung akzeptiert. In dieser Funktion wird die Dateiübertragung gestartet. Die `onDenied()` Funktion wird aufgerufen, wenn der Benutzer die Dateiübertragung ablehnt. In dieser Funktion wird die Dateiübertragung abgebrochen. Im folgenden Code-Ausschnitt ist exemplarisch die `onAccepted()` Funktion gezeigt.

```
private val fileActionReceiver = FileActionReceiver(
    onAccepted = { id ->
        val fileTransfer = pendingTransfers[id]
            ?: return@FileActionReceiver
        fileReceiveManagerScope.launch {
            fileTransfer.receiveFile(id)
            pendingTransfers.remove(id)
        }
    },
    // ...
)
```

receiveFile() Wie auch beim Senden der Datei, wird die Dateiübertragung in Chunks empfangen. Der Fortschritt der Dateiübertragung wird in der Benachrichtigung angezeigt. Die Datei wird in Chunks empfangen und gespeichert.

5.2.4 Unter Windows

Abbildung 44: UML LinkWaveSender

Nach der Auswahl der zu teilenden Dateien und des Empfängers initiiert LinkWave den Übertragungsprozess durch das Senden eines spezifischen 'Command' an das Empfängergerät. Dieser Vorgang wird durch den `LinkWaveSender` realisiert.

Commands Senden Der `LinkWaveSender` nutzt einen SSL Netzwerkstream, um den Command an den Empfänger zu senden.

```
public static async Task SendCommandAsync(...)
{
    // Serialisierung und Versendung des Commands
    LinkWaveCommandRequest command =
    new LinkWaveCommandRequest(requestType);
    var serializedCommand = DataSerializer.SerializeObject(command)
       ;
    await stream.WriteAsync(commandBytes, 0, commandBytes.Length);
    ...
}
```

Empfang und Verarbeitung des Commands

Auf der Empfängerseite nimmt der `LinkWaveReceiver` den Command entgegen. Diese Klasse ist für das Warten und Empfangen von eingehenden Befehlen und deren Verarbeitung und Weiterleitung an die Klasse `RequestSwitcher` zuständig. Die Methode StartLoopAsync initiiert eine asynchrone Schleife, die auf eingehende Verbindungen wartet. Die Methode verwendet SSLStream, um eine sichere Verbindung zu gewährleisten. `SSLStream` ist eine Klasse, die eine sichere Kommunikation über SSL (Secure Sockets Layer) oder TLS (Transport Layer Security) ermöglicht. Sie wird verwendet, um eine sichere Verbindung zwischen zwei Geräten herzustellen. [10]

LinkWaveReceiver.cs

```
public async Task StartLoopAsync(...)
{
    while (true)
    {
        // SSL Handshake
        SslStream sslStream = await
        SSL.AcceptAndAuthenticateClientAsync(..., certificate);
        _ = HandleClient(sslStream);
    }
}
```

Sobald eine Verbindung hergestellt ist, übernimmt die Methode HandleClient die Verarbeitung der Befehle.

```
private async Task HandleClient(SslStream sslStream)
{
    var bytesRead = await sslStream.ReadAsync(commandBytes,...);
    _ = RequestSwitcher.Handle(linkWaveCommandRequest
    ,sslStream,_uIFunctionStore);
    // Error Handling wurde ausgelassen
}
```

Request Behandeln

Die RequestSwitcher Klasse ist eine Schaltstelle für die Weiterleitung eingehender Befehle. Sie analysiert den Typ jeder Anfrage und delegiert sie an die entsprechenden Handler, die für die spezifische Bearbeitung zuständig sind. In dem Fall aktiviert der `RequestSwitcher` den `FileSharingHandler`, der den eigentlichen File-Sharing Prozess steuert.

```
public static async Task Handle(...)
{
    switch (commandRequest.RequestType)
    {
        case RequestTypes.FILE_SHARE:
        await FileSharingHandler
        .HandleAsync(networkStream,uIFunctionStore);
        break;
        // Andere Anfragen bzw. RequestTypes
    }
    // Error Handling wurde ausgelassen
}
```

File Sharing Handlers

Die 'FileReceiver' Klasse spielt eine wesentliche Rolle im Prozess des Dateiempfangs. Sie verwaltet die Annahme, Überprüfung und das Speichern von Dateien, die von einem anderen Gerät gesendet werden.

ReceiveFileInformation Diese Methode liest die eingehenden Daten aus dem Network Stream (verschickte Informationen), die Informationen über die zu empfangende Datei enthalten. Diese Informationen werden deserialisiert und in einem FileShareRequest Objekt gespeichert. Das FileShareRequest Objekt enthält Details wie den Dateinamen und die Dateigröße.

```
public async Task<FileShareRequest> ReceiveFileInformation()
{
    // Empfangen der Dateiinformationen
    _sharedFile = DataSerializer.
    DeserializeObject<FileShareRequest>(...)!;
    return _sharedFile;
    // Error Handling wurde ausgelassen
}
```

SendFileShareResponse Nachdem die Dateiinformationen empfangen und den Benutzer:innen angezeigt wurden, ermöglicht diese Methode dem Empfänger, eine Antwort zu senden, ob die Datei akzeptiert oder abgelehnt wird. Diese Entscheidung wird als FileShareResponse Objekt serialisiert und über den Netzwerk Stream an den Sender zurückgeschickt.

```
public async Task SendFileShareResponse(bool isAccepted)
{
    // Kontrolliere ob die Datei akzeptiert oder abgelehnt wurde
    var responseObject =
    new FileShareResponse(isAccepted ? FileShareResponseType.ACCEPT
    : FileShareResponseType.DECLINE);
    await _networkStream
    .WriteAsync(responseBytes, 0, responseBytes.Length);
    // Error Handling wurde ausgelassen
}
```

ReceiveFile Diese Methode ist für den eigentlichen Dateiempfang verantwortlich. Sie liest die Datei Bytes und schreibt sie in den Zielordner. Der Fortschritt wird in Prozent berechnet und an das User Interface übermittelt.

```
public async Task ReceiveFile(string fileSavePath)
{
    while (bytesLeftToRead > 0 && !cancelled)
    {
        ...
        var bytesRead = await _networkStream.ReadAsync(buffer, ...)
            ;
        if (bytesRead == 0) break;
        await fileStream.WriteAsync(buffer, 0, bytesRead);
        bytesLeftToRead -= bytesRead;
        // Aktualisierung des Fortschritts
    }
    // Error Handling wurde ausgelassen
}
```

Dateien Senden

Die FileSender-Klasse ist für die Initiierung und Durchführung der Dateiübertragung an einen Empfänger verantwortlich. Diese Klasse verwaltet mehrere Schritte des Übertragungsprozesses, von der Übermittlung der Dateiinformationen bis zum eigentlichen Senden des Dateiinhalts.

ReceiveAcknowledgement Diese Methode wartet auf eine Bestätigungsantwort (Acknowledgement) vom Empfänger, dass dieser bereit ist, die Datei zu empfangen. Sie liest die Antwort und kontrolliert, ob eine positive Bestätigung (ACK) vorliegt.

SendFileInformation Mit Hilfe der Methode ReceiveFileInformation, die bereits vorgestellt wurde, werden Dateiinformationen empfangen. Die SendFileInformation-Methode ist das Gegenstück und sendet die Dateiinformationen an den Empfänger.

In diesem Schritt sendet der FileSender Informationen über die zu sendende Datei, einschließlich des Dateinamens, der Größe und eines Hash-Werts für die Überprüfung der Integrität. Diese Daten werden serialisiert und über den networkStream an den

Empfänger gesendet. Diese Methode unterstützt sowohl Einzeldatei- als auch Mehrfachdateiübertragungen.

ReceiveFileShareResponse Nach dem Senden der Dateiinformationen wartet der Sender auf eine Antwort vom Empfänger, ob die Dateiübertragung akzeptiert wurde. Diese Methode liest die Antwort und prüft, ob die Dateiübertragung fortgesetzt werden kann.

SendFileContent Diese Kernmethode ist für das Senden der eigentlichen Dateiinhalte verantwortlich. Sie öffnet die Datei, liest sie in Blöcken und sendet diese Blöcke über den Network Stream zum Empfänger. Während der Übertragung wird optional eine Fortschrittsanzeige aktualisiert, um den Fortschritt der Übertragung zu verfolgen.

```
public async Task SendFileContent(string filePath)
{
    // Schleife zum Senden der Dateidaten.
    while ((bytesRead =
    await fileStream.ReadAsync(buffer, 0, buffer.Length)) > 0)
    {
        await _networkStream
        .WriteAsync(buffer, 0, bytesRead);
        transferredBytes += bytesRead;
        var bytes = transferredBytes;
    }
    // Error Handling wurde ausgelassen
}
```

5.3 Zwischenablage

Die Zwischenablage ist eine weitere sehr wichtige Funktion von LinkWave. Sie ermöglicht es, Texte und Bilder zwischen verschiedenen Geräten zu teilen.

Diese Funktion soll das nahtlose Wechseln zwischen verschiedenen Geräten während der Arbeit ermöglichen. So kann man beispielsweise eine Textnachricht auf einem Computer verfassen und anschließend über ein Telefon versenden. Man kann auch Bilder auf einem Telefon kopieren und in ein Dokument auf einem Computer einfügen.

Die Zwischenablage-Funktion ist auf allen Betriebssystemen zumindest zum Teil verfügbar. Auf Windows, macOS und Android ist es möglich, Texte und Bilder zu kopieren und zu empfangen. Auf iOS ist es möglich, Texte und Bilder zu empfangen, jedoch können nur Texte und nicht Bilder kopiert werden. Auf den mobilen Betriebssystemen wird die Zwischenablage-Funktion zudem über die Share-Funktion realisiert. Das bedeutet, dass man auf Mobilgeräten nicht die Zwischenablage-Funktion direkt verwenden kann, sondern nur einen extra Button im Textauswahl Menü benutzen kann.

5.3.1 Funktionsweise

Wie bereits beim File Sharing-Teil erwähnt, verwendet LinkWave eine Kombination aus Discovery-Protocol und TLS, um Geräte automatisch zu finden und Daten zwischen zwei Geräten verschlüsselt zu teilen. Nachdem Device A (Sender) über Bonjour Discovery ein kompatibles Gerät gefunden und eine gesicherte Verbindung über TLS (Transport Layer Security) aufgebaut hat, erfolgen folgende Schritte:

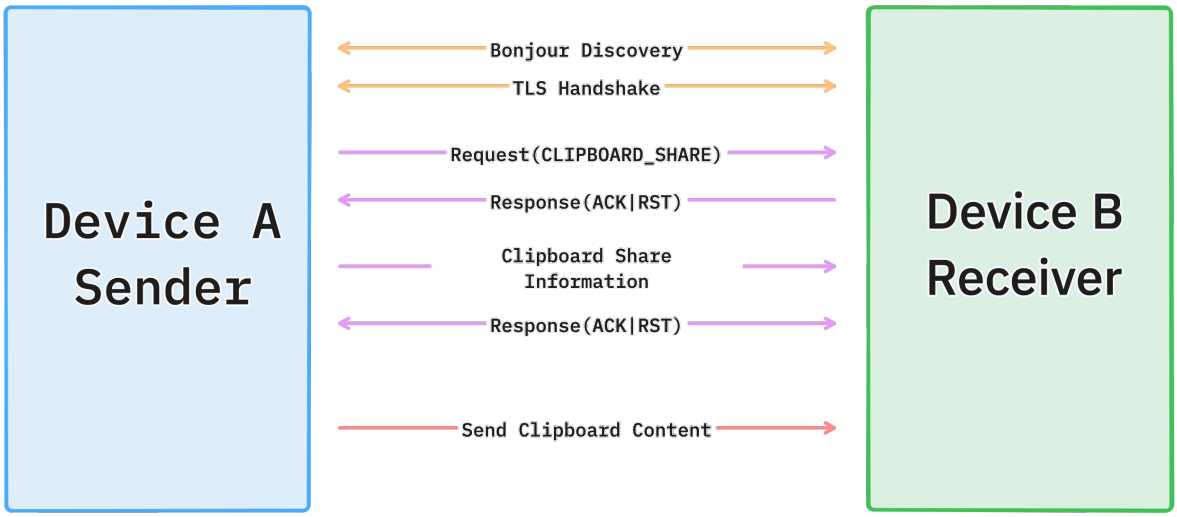


Abbildung 45: Clipboard Sharing

1. **Request (CLIPBOARD_SHARE):** Device A sendet eine Anfrage an Device B (Empfänger), um seine Absicht mitzuteilen und den Inhalt der Zwischenablage freizugeben.

2. **Response (ACK|RST):** Device B antwortet mit ACK, um eine erfolgreiche Anforderung zu bestätigen, oder mit RST, um sich zurückzusetzen, wenn ein Fehler auftritt.

3. **Send Clipboard Information:** Device A sendet nun die Informationen zur Freigabe der Zwischenablage. Diese Informationen können Details darüber enthalten, wie der Inhalt der Zwischenablage formatiert und übertragen werden soll.

4. **Response (ACK|RST):** Device B sendet seinerseits eine Antwort, um zu bestätigen, dass die Informationen über die gemeinsame Nutzung der Zwischenablage empfangen wurden.

5. **Send Clipboard Content:** Zum Schluss sendet das Gerät A den tatsächlichen Inhalt der Zwischenablage an das Gerät B.

5.3.2 Unter iOS und macOS

Da sich macOS und iOS die Codebase für Netzwerkzugriffe teilen, ist die Zwischenablage für iOS und macOS zusammen implementiert. Apple ermöglicht den Zugriff auf die Zwischenablage über die NSPasteboard Klasse auf macOS und die UIPasteboard Klasse auf iOS. Diese Klassen ermöglichen es, Texte und Bilder in die Zwischenablage zu kopieren und aus der Zwischenablage zu lesen.

Lesen der Zwischenablage Um die Zwischenablage auf macOS zu lesen wird ein Timer verwendet, der alle 2 Sekunden nachschaut, ob sich der Inhalt der Zwischenablage geändert hat. Die NSPasteboard.general.changeCount Property wird bei jeder Änderung der Zwischenablage erhöht. Wenn sich der Wert der Property ändert, weiß man, dass sich der Inhalt der Zwischenablage geändert hat. Um zu sehen, ob sich der Wert ändert speichert man den aktuellen Wert in einer Variable und vergleicht ihn mit dem Wert der Property. Wenn sich die Werte unterscheiden, hat sich der Inhalt der Zwischenablage geändert.

```
if currentChangeCount != NSPasteboard.general.changeCount {
    currentChangeCount = NSPasteboard.general.changeCount
    // Inhalt der Zwischenablage hat sich geaendert
}
```

Wenn sich die Zwischenablage geändert hat wir der Typ des Inhalts überprüft. Der Typ besteht aus einem Enum, in dem die möglichen Typen und Werte der Zwischenablage gespeichert sind. Der Typ wird dann verwendet, um den jeweiligen Inhalt der Zwischenablage zu lesen.

```
let type = NSPasteboard.general.availableType(
    from: [.string, .URL, .fileURL, .png, .tiff])
switch type {
  case .string:
    // Text aus Zwischenablage lesen
  case .png:
    // Bild aus Zwischenablage lesen
} // ... Andere Typen
```

Danach wird der Inhalt nach der in Kapitel 5.3.1 beschriebenen Methode an die Geräte des eigenen Accounts gesendet. Hierfür werden die in Kapitel 5.2.2 beschriebenen Methoden `NWConnection.send()` und `NWConnection.receive()` verwendet.

Schreiben in die Zwischenablage Das Schreiben in die Zwischenablage ist eine weitere mögliche Funktion, mit der im in Kapitel 5.1.2 beschriebenen New Connection Handler eine Verbindung gestartet werden kann. Hier wird jedoch durch das SSL Zertifikat sichergestellt, dass nur Geräte des eigenen Accounts auf die Zwischenablage zugreifen können. Man muss aufgrund leichter Unterschiede zwischen macOS und iOS zwei Compiler-Direktiven verwenden, um den Code für beide Betriebssysteme zu kompilieren.

```
#if os(macOS)
    let pasteBoard = NSPasteboard.general
    pasteBoard.clearContents()
    // ...
#elseif os(iOS)
    let pasteBoard = UIPasteboard.general
    pasteBoard.items = []
    // ...
#endif
```

Im ersten Schritt muss alter Inhalt aus der Zwischenablage gelöscht werden. Danach wird der neue Inhalt in die Zwischenablage geschrieben. Basierend auf dem Typ des Inhalts wird dieser (in der `data` Variable gespeichert) in die Zwischenablage geschrieben.

```
switch type { // auf macOS
  case .TEXT:
    pasteBoard.setString(
      String(data: data, encoding: .utf8),
      forType: .string
    )
  case .IMAGE:
    pasteBoard.setData(data, forType: .png)
}
```

```
switch type { // unter iOS
  case .TEXT:
    pasteBoard.string = String(data: data, encoding: .utf8)
  case .IMAGE:
    pasteBoard.image = UIImage(data: data)
}
```

Der Typ des Inhalts wird wie in Kapitel 5.3.1 beschrieben über die TCP Verbindung empfangen und in die `ClipboardShareRequest` Klasse deserialisiert. Diese Klasse enthält den Typ des Inhalts und die Größe in Bytes des Inhalts.

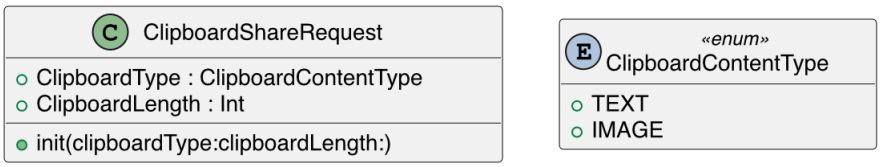


Abbildung 46: UML von ClipboardShareRequest

Die `data` Variable ist ein Objekt der gleichnamigen `Data()`. Sie enthält die Bytes des Inhalts der Zwischenablage. Diese Bytes sind im Falle von Texten die UTF-8 codierten Bytes des Textes und im Falle von Bildern die Bytes des Bildes. Hierbei ist es egal, ob es sich um ein PNG oder ein JPEG Bild handelt, da die Zwischenablage auf macOS und iOS beide Formate unterstützt und JPEGs einfach als PNGs sieht. Die Zwischenablagen auf anderen Betriebssystemen unterstützen diese Ambiguität ebenfalls. Deshalb sind als Typen nur `.TEXT` und `.IMAGE` definiert.

5.3.3 Unter Android

Zwischenablage teilen

Das Teilen der Zwischenablage funktioniert im Prinzip so wie das Teilen von Dateien. Anstatt eines Bildes oder einer Datei wird ein Text übertragen. Der Prozess ist in Abbildung 47 dargestellt.

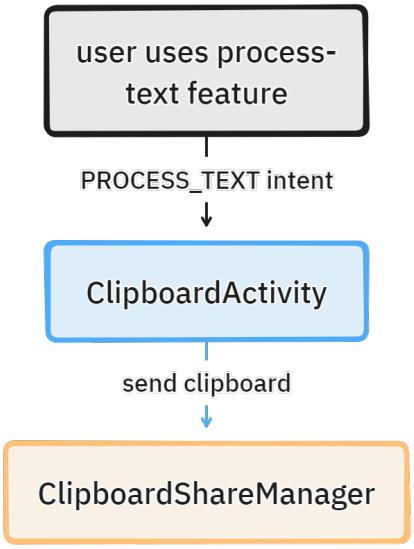


Abbildung 47: Teilen der Zwischenablage

Da seit Android 10 nicht mehr auf die Zwischenablage zugegriffen werden kann, wird das `PROCESS_TEXT` Feature von Android verwendet. Dies erlaubt es, einen markierten Text an eine App zu senden, die diesen Text dann weiterverarbeiten kann. Im Fall der LinkWave App wird bei einem `PROCESS_TEXT` Intent die ClipboardActivity geöffnet. Sie leitet den Text an des `ClipboardViewModel` weiter, das mittels des `ClipboardShareManager` den Text an alle Geräte sendet.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val devicesState = devicesViewModel.state.value

    intent?.let { i ->
        val selectedText = i.getStringExtra(Intent.
            EXTRA_PROCESS_TEXT)

        clipboardViewModel.shareClipboardText(selectedText,
            devicesState.scannedDevices)
    }

    finish()
}
```

In obigem Code-Ausschnitt wird der markierte Text aus dem Intent gelesen und an das `ClipboardViewModel` weitergeleitet. Die `finish()` Funktion beendet die Activity nachdem der Text weitergeleitet wurde. Für den Benutzer ist dieser Vorgang unsichtbar.

```
fun shareClipboardText(text: String, devices: List<LinkWaveDevice>)
    {
    for (device in devices) {
        viewModelScope.launch {
            clipboardShareManager.shareClipboardText(text, device)
        }
    }
}
```

Der Text wird dann als `ByteArray` über den `NetworkStream` an die Geräte gesendet. Der Empfänger kann den Text dann in die Zwischenablage kopieren.

```
val textBytes = text.toByteArray()
outputStream.write(textBytes)
```

Zwischenablage empfangen

Wie auch das Empfangen von Dateien, soll das Empfangen von Texten und Bildern in der Zwischenablage auch im Hintergrund funktionieren. Der Empfangsprozess wird in dem `LinkWaveService` (siehe 5.5.3) durchgeführt und wird in Abbildung 48 dargestellt.

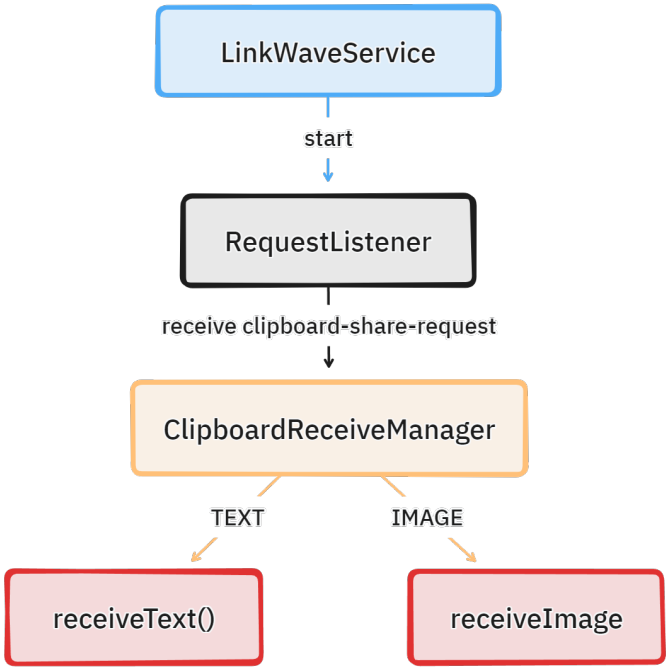


Abbildung 48: Prozess der Zwischenablageübertragung auf Android

Wenn der RequestListener eine `CLIPBOARD_SHARE` Anfrage empfängt, wird diese an den `ClipboardReceiveManager` weitergeleitet. Der ClipboardReceiveManager bekommt den `NetworkStream` mitgegeben, über den er dann den Typ des Inhalts übermittelt bekommt. Bei einem Text wird dieser dann mit der `receiveClipboardText` Funktion in die Zwischenablage kopiert. Bei einem Bild wird dieses mit der `receiveClipboardImage` Funktion in die Zwischenablage kopiert.

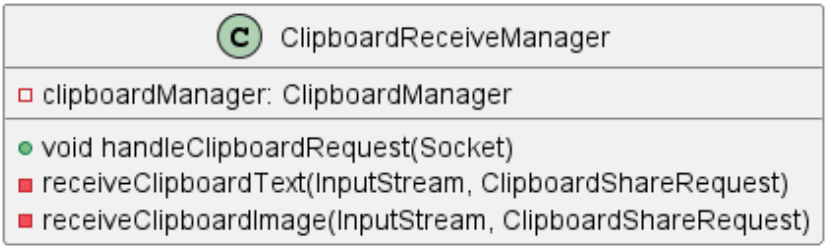


Abbildung 49: UML von ClipboardReceiveManager

receiveClipboardText() Diese Funktion liest den Text aus dem `NetworkStream` und kopiert ihn in die Zwischenablage. Folgender Code-Ausschnitt zeigt die Implementierung.

```
val textBytes = ByteArray(clipboardShareRequest.length)
val textBytesRed = inputStream.read(textBytes)
text = String(textBytes.sliceArray(0 until textBytesRed))

clipboardManager.setPrimaryClip(
    ClipData.newPlainText("linkwave_text", text)
)
```

receiveClipboardImage() Diese Funktion liest das Bild aus dem `NetworkStream` und kopiert es in die Zwischenablage. Wie auch bei der Dateiübertragung wird das Bild in Chunks übertragen. Folgender Code-Ausschnitt zeigt die Implementierung.

```
val uri =
    FileProvider.getUriForFile(
        androidContext,
        "${androidContext.packageName}.fileprovider",
        tempFile
    )
val clipData = ClipData.newUri(androidContext.contentResolver, "
   LinkWave image", uri)
clipboardManager.setPrimaryClip(clipData)
```

Der Content Provider ist notwendig um Dateien in andere Apps zu übertragen und wird in der `AndroidManifest.xml` Datei definiert. Die Konfiguration des `FileProvider` ist in foglendem Code-Ausschnitt gezeigt.

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths" />
</provider>
```

Der `FileProvider` erlaubt es Apps auf die geteilten Bilder zuzugreifen. Welche Dateien geteilt werden dürfen wird in der `file_paths.xml` Datei definiert. In diesem Fall wird nur der temporäre Ordner definiert.

5.3.4 Unter Windows

Clipboard Sharing funktioniert im Prinzip wie File Sharing, allerdings mit einigen Unterschieden. Hier kann der Benutzer nicht bestätigen, ob er den Inhalt der Zwischenablage erhalten möchte oder nicht. Nach dem Kopieren des Inhalts durch den Benutzer wird dieser automatisch nur an die Geräte gesendet, die mit demselben Konto eingeloggt sind. Bei Clipboard Sharing werden die Anfragen genau wie bei File Sharing verarbeitet und weitergeleitet.
Clipboard Sharing unterteilt sich in zwei Klassen: ClipboardSender und ClipboardReceiver.

ClipboardReceiver

ReceiveTypeAsync Diese Methode ist der erste Schritt im Empfangsprozess und liest die Art des geteilten Inhalts (z.B. Text oder Bild) aus dem Network Stream. Sie deserialisiert die empfangenen Daten zu einem ClipboardShareRequest Objekt, das Informationen über die Art und Länge des Inhalts enthält. Nach erfolgreichem Empfang und Deserialisierung der Inhaltsinformationen sendet die Methode eine Bestätigungsantwort (ACK) zurück an den Sender, um die Bereitschaft zum Empfang der eigentlichen Inhaltsdaten zu signalisieren.

```
public async Task ReceiveTypeAsync()
{
    _ = await _networkStream.ReadAsync(receiveTypeBytes);
    // Deserialisierung der empfangenen Daten
    await LinkWaveSender.SendCommandAsync(equestTypes.ACK,...);
}
```

ReceiveClipboardContentAsync Diese Methode wartet auf die Übertragung des eigentlichen Clipboard Inhalts, nachdem sie den Inhaltstyp empfangen und bestätigt hat. Sie ruft ReceiveClipboardContentBytes auf, um die Inhaltsbytes zu empfangen. Je nach Typ des Inhalts werden unterschiedliche Aktionen ausgeführt

```
public async Task ReceiveClipboardContentAsync()
{
    switch (_shareRequest.ClipboardType)
    {
        case ClipboardContentType.Text:
            _=_uIFunctionStore.SetLastClipboardReceived(text);
            _=_uIFunctionStore.SetClipboardText(text);
            break;
        case ClipboardContentType.Image:
            // Empfangen der Bilddaten
            _=_uIFunctionStore.SetLastClipboardReceived(checksum);
            _=_uIFunctionStore.SetClipboardImage(bytes);
            break;
    }
}
```

ReceiveClipboardContentBytes Nach dem Aufruf von ReceiveClipboardContent liest diese Methode den gesendeten Inhalt Byte für Byte ein und speichert ihn in einem Byte-Array. Diese Methode stellt sicher, dass die gesamte Nachricht, unabhängig von ihrer Größe, effizient und vollständig empfangen wird.

```
    private async Task<byte[]> ReceiveClipboardContentBytes(int
       length)
    {
        ...
        while (bytesRead < length)
        bytesRead += await _networkStream.ReadAsync(buffer, ...));
        return buffer;
    }
```

Schreiben in die Zwischenablage Das Lesen und Schreiben in die Zwischenablage wird auf Windows unter C# von der `System.Windows.Clipboard` Klasse übernommen. Diese Klasse enthält statische Methoden, um mit der Zwischenablage zu interagieren. Um Text zu setzen wird ganz einfach die `SetText()` Methode verwenden.

```
private async Task SetClipboardTextAsync(string text)
{
    System.Windows.Clipboard.SetText(text);
}
```

Beim Setzen von Bildern in die Zwischenablage muss man eine etwas kompliziertere Variante wählen. Es gibt zwar eine `SetImage()` Methode, diese funktioniert jedoch nur mit Bitmaps. Das bedeutet, dass jegliche Bildformate mit transparentem Hintergrund diesen verlieren. Um dies zu umgehen, muss man ein eigenes `DataObject` erstellen. Diesem wird das gewöhnliche Bitmap Feld hinzugefügt, um Kompatibilität mit älteren Programmen zu gewährleisten. Nun wird auch noch ein `PNG` Feld erstellt. Diesem wird das Bild in Form eines Byte-Arrays hinzugefügt. Dieses DataObject wird dann in die Zwischenablage kopiert. So können auch Bilder mit transparentem Hintergrund in die Zwischenablage kopiert werden, solange das empfangende Programm das PNG Feld auslesen kann. Programme wie Microsoft Word, Google Docs oder Photoshop können dies.

```
private async Task SetClipboardImageAsync(byte[] imageBytes)
{
    var dataObject = new System.Windows.DataObject();
    var memStream = new MemoryStream(imageBytes)
    var bitmapImage = new BitmapImage();
    bitmapImage.BeginInit();
    bitmapImage.StreamSource = memStream;
    bitmapImage.EndInit();
    dataObject.SetData(DataFormats.Bitmap, bitmapImage, true);
    dataObject.SetData("PNG", memStream, false);
    System.Windows.Clipboard.SetDataObject(dataObject, true);
}
```

ClipboardSender

Die Kommunikation beginnt mit dem Senden eines Initialisierungsbefehls, um den Prozess zu starten. Nach dem Senden wird auf eine Bestätigung (ACK) vom Empfänger gewartet. Diese Bestätigung stellt sicher, dass der Empfänger bereit ist, weitere Daten zu empfangen. Sobald die Bestätigung eingetroffen ist, sendet der Clipboard-Sender detaillierte Informationen über den Inhalt der Zwischenablage. Nach der zweiten Bestätigung beginnt die eigentliche Übertragung des Inhalts.

```
public async Task SendAsync(SslStream networkStream)
{
    // Senden des Initialisierungsbefehls
    while (_contentBytes.Length > 0)
    {
        var chunkSize = Math.Min(...);
        await networkStream.WriteAsync(_contentBytes, 0, chunkSize)
            ;
        _contentBytes = _contentBytes[chunkSize..];
    }
}
```

5.4 Benutzerkonto

Das LinkWave Benutzerkonto ist ein wichtiger Bestandteil der Anwendung, da es die Authentifizierung und Autorisierung der Benutzer ermöglicht.
Da LinkWave Geräte sich automatisch verbinden, ist es wichtig, dass keine unbefugten Geräte auf Daten zugreifen können. Andere Alternativen wie KDE Connect erwarten, dass Benutzer:innen Geräte manuell verbinden. Um die Benutzung zu vereinfachen, wurde bei LinkWave auf eine manuelle Verbindung verzichtet. LinkWave Geräte verbinden sich automatisch, wenn sie sich im selben Netzwerk befinden. Damit wird jedoch eine Sicherheitslücke eröffnet, da sich unbefugte Geräte ebenfalls im selben Netzwerk befinden können. Um dies zu verhindern, wird ein Benutzerkonto benötigt.
Manche Funktionen wie die Zwischenablage sind nur für angemeldete Benutzer:innen verfügbar, da sie viel Einfluss über die Geräte haben. Andere Funktionen sind auch ohne Anmeldung verfügbar.
LinkWaveApp Benutzerkonten werden entweder in den jeweiligen Apps oder auf dem Web-Dashboard erstellt. Man hat die Möglichkeit sich mit einer E-Mail Adresse und einem Passwort zu registrieren oder sich mit einem Google Konto anzumelden.

5.4.1 Funktionsweise

Die Benutzerverwaltung ist ein wichtiger Bestandteil der API, da sie die Authentifizierung und Autorisierung der Benutzer ermöglicht.
Die Implementierung umfasst die Verwaltung von Benutzerkonten, die Authentifizierung über JSON Web Tokens (JWT) und die Integration von Google OAuth für die Authentifizierungsdienste.
Darüber hinaus wird die Zwei-Faktor-Authentifizierung (2FA) unterstützt, um die Sicherheit der Benutzerkonten zu erhöhen.

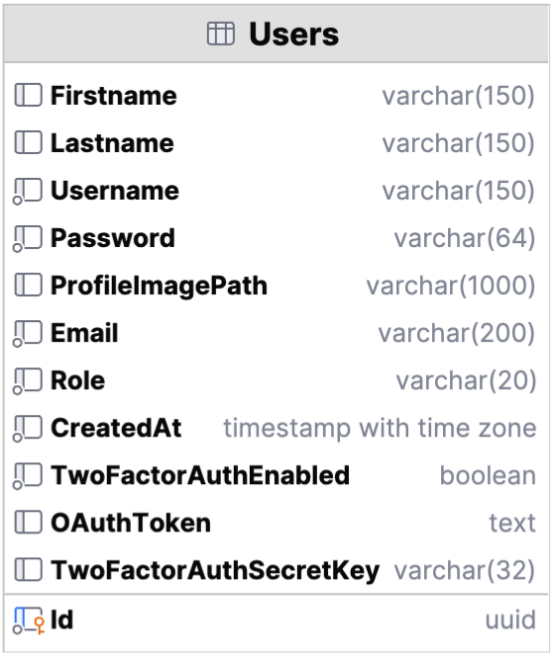


Abbildung 50: Users Tabelle

Die 'Users'-Tabelle im PostgreSQL-Datenbanksystem von LinkWave beinhaltet folgende Felder:

Datenbankstruktur

- **Firstname:** Der Vorname des Benutzers.
- **Lastname:** Der Nachname des Benutzers.
- **Username:** Der eindeutige Benutzername für den Login.

- **Password:** Das verschlüsselte Passwort des Benutzers.
- **ProfileImagePath:** Der Dateipfad zum Profilbild des Benutzers.
- **Email:** Die E-Mail-Adresse des Benutzers.
- **Role:** Die Rolle des Benutzers, welche die Zugriffsrechte definiert.
- **CreatedAt:** Das Erstellungsdatum und die Uhrzeit des Benutzerkontos.
- **TwoFactorAuthEnabled:** Ein boolean, der angibt, ob die Zwei-Faktor-Authentifizierung aktiviert ist.
- **OAuthToken:** Ein Token für OAuth-Authentifizierungsverfahren.
- **TwoFactorAuthSecretKey:** Ein secret key für die Zwei-Faktor-Authentifizierung.
- **Id:** Die eindeutige ID des Benutzerkontos.

Jedes Feld dient einem spezifischen Zweck und ist entscheidend für die Authentifizierung, Autorisierung und Verwaltung der Benutzerinformationen. Die Datenbankstruktur unterstützt starke Sicherheitsmaßnahmen und sorgt dafür, dass sensible Daten, wie Passwörter und Authentifizierungsschlüssel, sicher gespeichert werden. Die UUIDs garantieren die Einzigartigkeit jedes Datensatzes, während die E-Mail-Adressen und Profilbilder zur persönlichen Identifikation der Benutzer beitragen. Die Rollenverteilung ist essentiell für das Berechtigungsmanagement im System.

Endpunkte

Im LinkWave-Backend sind einige Endpunkte von zentraler Bedeutung für die Benutzerverwaltung, da sie die Grundlage für Sicherheits- und Funktionsaspekte bilden. Unter den vorhandenen Endpunkten sind fünf Endpunkte von besonderer Bedeutung, da sie für grundlegende Funktionen wie die Anmeldung und Registrierung von Benutzern und die Implementierung und Verwaltung der Two Factor Authentication unerlässlich sind.

LoginWithEmail

Die Funktion *LoginWithEmail* ermöglicht es Benutzer:innen, sich mit ihrer E-Mail Adresse und ihrem Passwort anzumelden. Bei erfolgreicher Authentifizierung wird ein JWT generiert und zurückgegeben, welcher für spätere Anfragen als Authentifizierungstoken dient.

LoginOAuthGoogle

LoginOAuthGoogle behandelt die Anmeldung über Google OAuth, was den Nutzern ermöglicht, sich mit ihrem Google-Konto anzumelden. Nach der Authentifizierung durch Google wird ebenfalls ein JWT zur weiteren Verwendung ausgestellt.

Register

Die *Register*-Funktion ist zuständig für die Registrierung neuer Nutzer im System. Benutzer:innen müssen grundlegende Informationen wie E-Mail, Passwort und ihren Namen bereitstellen, um ein neues Konto zu erstellen.

GenerateQrCodeUri

Mit *GenerateQrCodeUri* können Benutzer:innen, die Zwei-Faktor-Authentifizierung aktivieren, indem sie einen QR-Code generieren, der mit einer Authenticator-App gescannt wird. Dies fügt eine zusätzliche Sicherheitsebene zu ihrem Konto hinzu.

ValidateTwoFactorAuthentication

Die Funktion *ValidateTwoFactorAuthentication* wird verwendet, um den durch die Authenticator-App generierten Code zu validieren. Dieser Schritt ist erforderlich, um den Zugang zum System zu gewähren, wenn die Zwei-Faktor-Authentifizierung aktiviert ist.
Diese Punkte werden in den folgenden Abschnitten näher erläutert.

JSON Web Token

Ein JSON Web Token (JWT) ist ein kompaktes und selbstständiges Mittel zur sicheren Übertragung von Informationen zwischen Parteien als JSON-Objekt. Die Informationen

sind zuverlässig, da sie digital signiert sind. JWTs können mit einem geheimen Schlüssel (mit dem HMAC-Algorithmus) oder einem öffentlichen/privaten Schlüsselpaar unter Verwendung von RSA oder ECDSA signiert werden. [14]

Aufbau eines JWT

- **Header:** Der Header besteht typischerweise aus zwei Teilen: dem Typ des Tokens, der JWT ist, und dem verwendeten Signaturalgorithmus, wie zum Beispiel HMAC SHA256 oder RSA.

- **Payload (Body):** Der Payload enthält die sogenannten Claims. Claims sind Aussagen über eine Entität (typischerweise den Benutzer) und zusätzliche Metadaten. Es gibt drei Typen von Claims: registrierte, öffentliche und private Claims.

- **Signature:** Um die Signatur zu erstellen, muss der kodierte Header und der kodierte Payload mit dem gegebenen Algorithmus aus dem Header signiert werden, welcher durch einen geheimen Schlüssel gesichert ist.

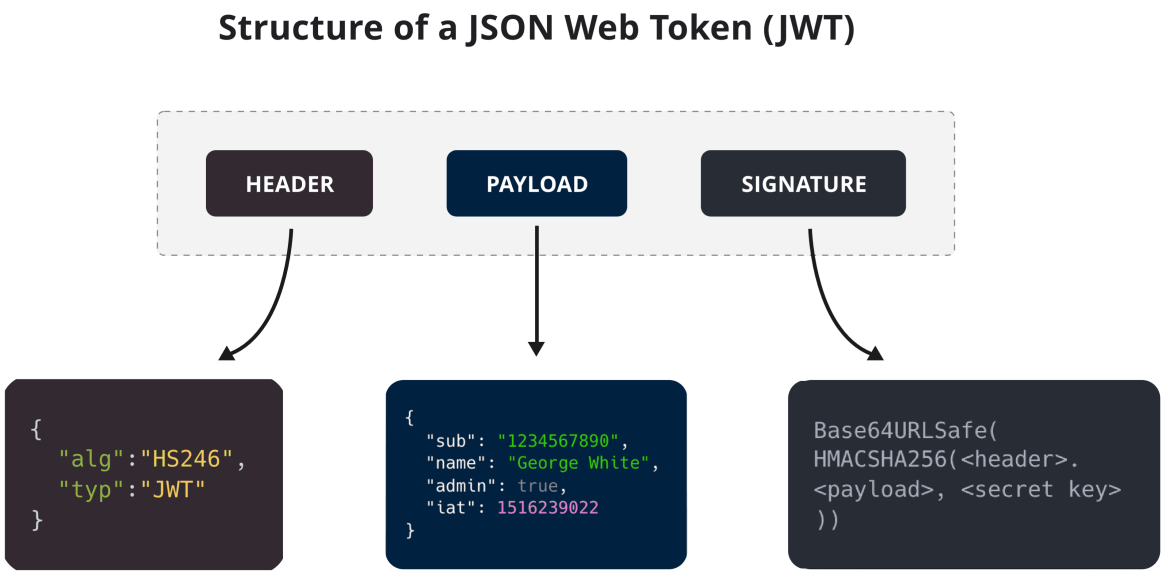


Abbildung 51: JWT Aufbau

Nutzung von JWTs Bei der Authentifizierung, wenn sich der Benutzer erfolgreich mit seinen Login-Daten anmeldet, gibt der Authentifizierungsserver einen JWT zurück. Der Client speichert diesen Token und sendet ihn bei folgenden Anfragen mit, üblicherweise im Authorization-Header, um den Server zu informieren, dass die Anfrage autorisiert ist.

Refresh Tokens Um die Benutzerfreundlichkeit zu verbessern und gleichzeitig die Sicherheit zu gewährleisten, werden Refresh Tokens verwendet. Ein Access Token hat in der Regel eine kurze Lebensdauer. Wenn es abläuft, würden Benutzer:innen ohne Refresh Token gezwungen sein, sich erneut anzumelden. Ein Refresh Token ermöglicht es, ein neues Access Token zu erhalten, ohne dass sich der Benutzer wieder anmelden muss, was eine nahtlose Benutzererfahrung und fortgesetzte Sicherheit ermöglicht.

Token Generierung

- Bei der Generierung eines neuen Tokens werden **Claims** für Benutzer:innen erstellt, einschließlich seiner ID, E-Mail, Rolle und des Token-Verfallsdatums.
- Ein **JwtSecurityToken** Ein JWT Security-Token wird generiert und enthält Issuer, Audience, Claims, Expire Date und SigningCredentials.
- Die **Signierungsinformationen** werden mit dem Sicherheitsschlüssel (SymmetricSecurityKey) und dem verwendeten Algorithmus, in diesem Fall HMAC SHA256, konstruiert.
- Der fertige Token wird dann zu einem String kodiert und zurückgegeben.

Refresh Token Generierung

- Ein Refresh Token wird generiert, indem eine 32-Byte lange zufällige Nummer erzeugt und zu einem Base64-String konvertiert wird.
- Dieses Token hat eine längere Lebensdauer als der Access Token und kann verwendet werden, um einen neuen Access Token zu erhalten, ohne dass sich Benutzer:innen authentifizieren müssen.

Sicherheit und Validierung

Durch die Verwendung von **Claims** und einer starken Verschlüsselung gewährleistet LinkWave, dass sowohl die Identität der Benutzer:innen als auch die Integrität der Token gesichert sind. Die generierten Tokens entsprechen den Sicherheitsbestimmungen und sind für die Dauer ihrer Gültigkeit verlässliche Authentifizierungsnachweise.

Google OAUTH

LinkWave integriert Google OAuth, um Benutzern eine nahtlose und sichere und einfache Anmeldung zu ermöglichen. Diese Integration verwendet spezifische Bibliotheken, um die Authentifizierung mit Google's Sicherheitsservern durchzuführen.

Was ist OAUTH?

Der OAuth-Flow von Google ist ein Prozess, der Anwendungen erlaubt, sicher auf die Google-Dienste im Namen von Benutzer:innen zuzugreifen. Im Folgenden wird die Struktur dieses Authentifizierungsmechanismus beschrieben und anhand eines Diagramms visualisiert.

Abbildung 52: Google Logo

OAuth ist ein offener Standard für Zugriffsberechtigungen, der es Benutzer:innen ermöglicht, privaten Inhalt wie Kontaktdaten sicher über HTTP-Dienste zu teilen. Es ermöglicht der Anwendung, im Namen von Benutzer:innen zu handeln, ohne dass das Passwort des des:derselben erforderlich ist. OAuth arbeitet mit Token, die als sichere

Authentifizierungsschlüssel dienen, und ermöglicht es, diese Zugriffsrechte zu spezifizieren und einzuschränken. Es ist die bevorzugte Methode für moderne API-Sicherheit und wird von den meisten großen Online-Dienstleistern unterstützt. [13]

Das OAuth-Flow

Anfrage der Erlaubnis Zu Beginn muss die Anwendung, die auf die Google-Dienste zugreifen möchte, von den Benutzer:innen die Erlaubnis einholen. Dies erfolgt über eine URL, die zu einer von Google bereitgestellten Seite führt, auf der die Benutzer:innen seine Zustimmung geben können.

Die Rolle der Anwendungsdaten Wichtige Komponenten in dieser Anfrage umfassen:

- **Client ID**: Eine einzigartige Kennung für die Anwendung, die den Zugriff beantragt.
- **Scope**: Definiert den Umfang des Zugriffs, den die Anwendung anfragt.
- **Redirect URI**: Eine vordefinierte Adresse, zu der Google den Benutzer nach der Genehmigung navigiert.

Zustimmung und Access Token Nach der Zustimmung der Benutzer:innen über die Google-Seite erhält die Anwendung einen Authorization Token, mit dem sie die Identität der Benutzer:innen bestätigen und im nächsten Schritt einen Access Token anfordern kann.

Zugriffstoken

- **Access Token**: Ermöglicht den Zugriff auf Google-Dienste für eine begrenzte Zeit.
- **Refresh Token**: Kann genutzt werden, um neue Access Tokens zu erhalten, ohne dass die Benutzer:innen sich erneut anmelden müssen.

Mit dem Access Token kann die Anwendung im Namen der Benutzer:innen agieren und auf die angeforderten Dienste wie Gmail, Google Calendar, Classroom usw. zugreifen.

Erneuerung des Zugriffs Nach Ablauf des Access Tokens kann die Anwendung den Refresh Token verwenden, um ein neues Access Token zu erhalten, ohne die Benutzer:innen zu stören.

Nachfolgend ein Diagramm zur Veranschaulichung des Ablaufs von Google OAuth:

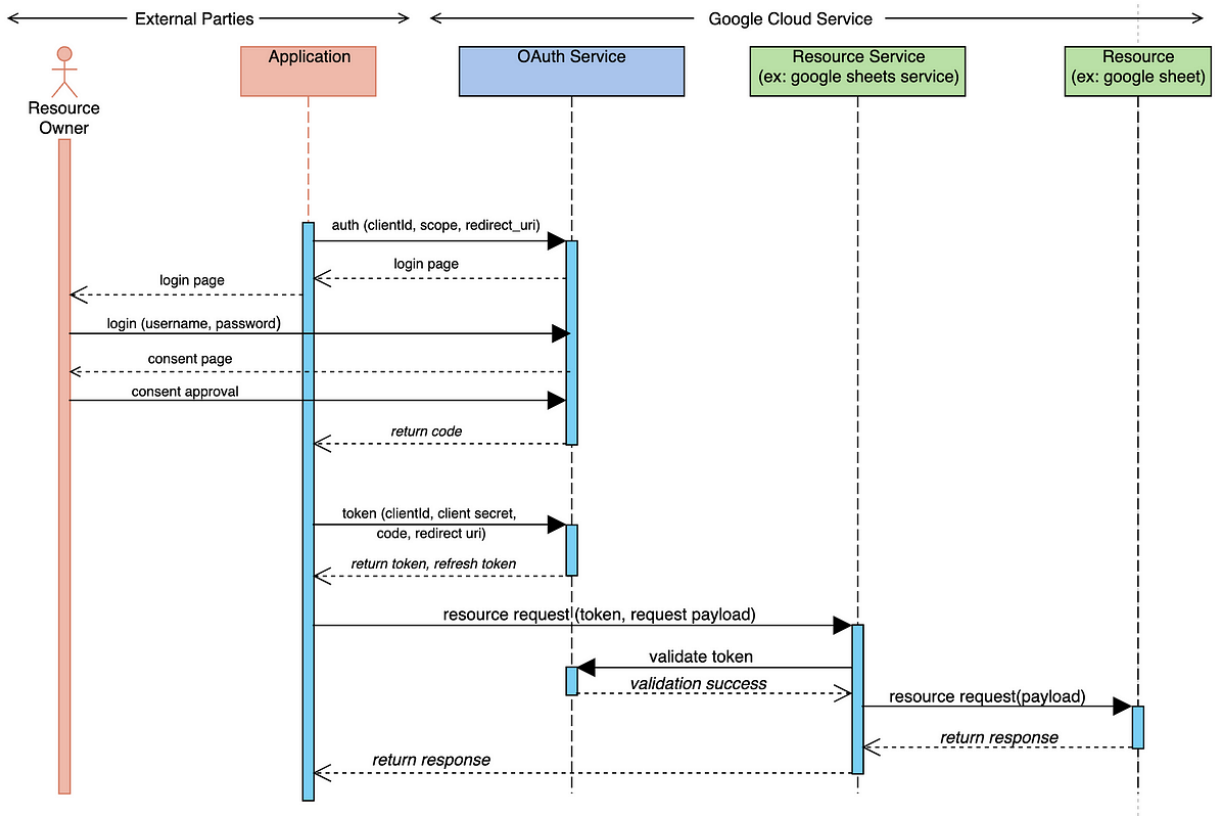


Abbildung 53: OAUTH Flow

Two Factor Authentication

Two Factor Authentication (2FA) stellt eine erweiterte Sicherheitsmethode dar, die über die traditionellen Benutzernamen und Passwörter hinausgeht, indem sie einen zweiten Authentifizierungsschritt verlangt. Dieser Schritt kann die Form von etwas, das der Nutzer besitzt, etwas, das der Nutzer weiß, oder etwas, das biometrisch einzigartig ist, annehmen. [15]

Two-Factor Authentifizierungsmethoden

In der Two-Factor Authentifizierung (2FA) gibt es verschiedene Methoden, um die Identität von Benutzer:innen zu bestätigen. Hierbei wird typischerweise ein One-Time Password (OTP) verwendet, das entweder zeitbasiert (TOTP) oder zählerbasiert (HOTP) ist. Diese Methoden nutzen kryptografische Funktionen wie HMAC, um sicherzustellen, dass jedes Passwort einzigartig und nicht reproduzierbar ist. [16]

Time-based One-Time Password (TOTP)

- TOTP ist ein Algorithmus, der ein temporäres Passwort generiert, das sich in regelmäßigen, kurzen Zeitintervallen ändert.
- Die Authentifizierungsserver und die Authenticator-App des Benutzers sind synchronisiert und verwenden die aktuelle Uhrzeit als Basis für das Passwort.
- TOTPs bieten einen guten Schutz gegen Replay-Angriffe, da selbst wenn ein Passwort abgefangen wird, dieses sehr schnell seine Gültigkeit verliert.

HMAC-based One-Time Password (HOTP)

- HOTP generiert Authentifizierungscodes, die auf einem geheimen Schlüssel und einem Zähler basieren.
- Der Server und das Authentifizierungsgerät des Benutzers halten beide den gleichen Zählerstand, der nach jeder Verwendung inkrementiert wird.
- HOTP-Token sind nicht zeitabhängig, was sie in Situationen ohne zuverlässige Zeitquelle nützlich macht.

Keyed-Hash Message Authentication Code (HMAC)

- HMAC ist ein spezieller Typ eines Message Authentication Codes (MAC), der eine kryptografische Hash-Funktion mit einem geheimen kryptografischen Schlüssel kombiniert.

- In 2FA-Systemen wird HMAC verwendet, um die One-Time Passwords zu generieren, die für TOTP und HOTP notwendig sind.
- Die Verwendung von HMAC sorgt für die Integrität und Authentizität der Nachrichten (in diesem Fall der Passwörter).

Two Factor Authentication in LinkWave

In LinkWave wurde Two Factor Authentication (2FA) im Backend implementiert, um die Sicherheit der Benutzerkonten zu erhöhen. Dieser Abschnitt beschreibt die technischen Schritte der Implementierung und wie Benutzer:innnen 2FA verwenden können.

Two Factor Authentication Flow

Um die 2FA zu aktivieren, generieren die Benutzer:innen über die Web, Desktop oder Mobile App einen QR-Code:

1. Benutzer:innen starten die 2FA Einrichtung im Benutzerkonto.
2. Ein eindeutiger, geheimer Schlüssel (secret key) wird serverseitig erstellt.
3. Mit dem Schlüssel wird ein QR-Code generiert, der die Identifikationsdaten für die 2FA-App der Benutzer:innen enthält.
4. Der Benutzer:innen scannen diesen QR-Code mit einer Authenticator-App, die auf dem mobilen Gerät installiert ist.
5. Die Authenticator-App speichert die Informationen und beginnt, Time-Based One-Time Password (TOTP) zu generieren.

Validierung der Zwei-Faktor-Authentifizierung

Bei jedem Anmeldeversuch müssen Benutzer:innen zusätzlich zum Passwort einen TOTP-Code angeben:

1. Benutzer:innen geben ihr Passwort und den von der Authenticator-Anwendung generierten TOTP-Code in die Anmeldeschnittstelle ein.
2. Der LinkWave-Server erhält den TOTP-Code und überprüft ihn mithilfe des gespeicherten geheimen Schlüssels (secret key).
3. Ist die Validierung erfolgreich, erhalten die Benutzer:innen Zugang zu ihrem Konto.

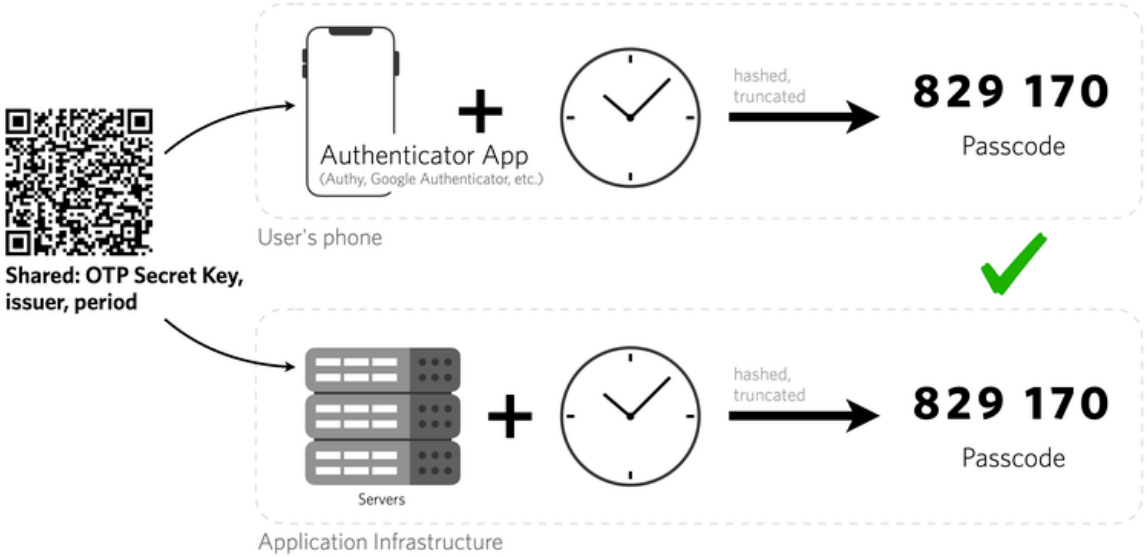


Abbildung 54: 2FA Flow

5.4.2 Unter iOS und macOS

Das Benutzerkonto wird auf iOS und macOS in der App verwaltet. Hier können sich Benutzer:innen registrieren, einloggen und ihr Profil bearbeiten. Es ist möglich sich mit einer Email Adresse und einem Passwort oder mit einem Google Konto anzumelden. Diese Varianten sind jedoch sehr unterschiedlich implementiert.

Email und Passwort

Das Anmelden und Registrieren mit Email und Passwort wird über eine GraphQL Verbindung zum LinkWave Server durchgeführt. Um Graphql zu verwenden, wir die Bibliothek `swift-graphql` verwendet. Die Dokumentation zu den verwendeten Funktionen ist auf der Webseite [2] zu finden. Diese Bibliothek ermöglicht es, GraphQL Queries und Mutations zu erstellen und zu senden. Außerdem unterstützt die Bibliothek das Generieren von Code aus GraphQL Schemas. Dies erleichtert die Verwendung von GraphQL in Swift. Das Schema wird mit folgendem Befehl generiert:

```
$ swift-graphql https://api.linkwave.org/graphql/ \
      -o generated.swift
```

Nun werden Funktionen und Klassen generiert, die man mit der `swift-graphql` Bibliothek verwenden kann. Um Benutzer:innen anzumelden wird eine `loginWithEmail` Anfrage geschickt.

```
let query = Objects.Mutation.loginWithEmail(
  email: email,
  password: password)
let request = URLRequest(url: linkWaveHelper.graphqlClient!)
  .querying(query)
let task = URLSession.shared.dataTask(with: request)
{ data, _, _ in
  guard let result = try? data?.decode(query) else { return }
  // Token speichern
}
```

In der `token` Variable wird nun der JWT gespeichert, welcher für die Authentifizierung weiterer Anfragen benötigt wird. Der Token wird außerdem in der Keychain gespeichert, um ihn auch nach einem Neustart der App zu behalten und trotzdem sicher zu speichern. Dies ist durch die `KeychainSwift` Bibliothek [3] möglich.

```
let token = result.data!
let keychain = KeychainSwift()
keychain.set(token, forKey: "JWT")
```

Das Registrieren funktioniert konzeptionell genau gleich wie das Anmelden, nur dass anstatt der `loginWithEmail` eine `register` Mutation geschickt wird. Diese Mutation erwartet zusätzlich noch den Benutzernamen, Vornamen und Nachnamen der Benutzer:innen.

Google Konto

Die Anmeldung mit einem Google Konto (OAuth) funktioniert auf iOS und macOS über die `GoogleSignIn` Bibliothek. Diese bietet eine einfache `signIn` Funktion, die den User im geöffneten Browserfenster durch den Anmeldeprozess führt. Nachdem die Benutzer:innen sich angemeldet haben, wird eine Callback Funktion aufgerufen, in der mithilfe des `accessToken` der JWT vom Server geholt wird.

```
GIDSignIn.sharedInstance.signIn(withPresenting: mainWindow)
{ signInResult, _ in
  guard let result = signInResult else { return }
  var accessToken = result.user.accessToken
  var query = Objects.Mutation.loginOAuthGoogleAccessToken(
    accessToken: accessToken.tokenString)
}
```

Der JWT wird nun wieder in der Keychain gespeichert, um ihn für weitere Anfragen zu verwenden.

5.4.3 Unter Android

Die Benutzerverwaltung auf Android wird durch die Android-App verwaltet. Hier können sich Benutzer:innen registrieren, und mit Email und Passwort anmelden. Dafür wird die GraphQL API des LinkWave Servers verwendet. Für die Kommunikation mit der GraphQL API wird die Bibliothek `Apollo-Android` verwendet. Diese Bibliothek ermöglicht es, GraphQL Queries und Mutations zu erstellen und zu senden. Außerdem unterstützt die Bibliothek das Generieren von Code aus GraphQL Schemas.

Email und Passwort

Um Benutzer:innen anzumelden wird eine `loginWithEmail` Anfrage an den Server geschickt. Diese Anfrage enthält die Email und das Passwort der Benutzer:innen. Die Antwort des Servers enthält den JWT Token, der für die Authentifizierung bei zukünftigen Anfragen verwendet wird. Die GraphQL Anfrage ist in folgendem Codebeispiel dargestellt.

```
mutation LoginWithEmail($email: String!, $password: String!) {
    loginWithEmail(email: $email, password: $password)
}
```

Aus dieser Mutation generiert Apollo-Android automatisch eine Klasse, welche die Anfrage enthält. Diese Klasse wird verwendet, um die Anfrage an den Server zu senden. Der Server antwortet mit dem JWT Token, der in den SharedPreferences gespeichert wird. Folgender Code zeigt, wie die Anfrage an den Server gesendet wird und die Antwort verarbeitet wird.

```
val response = apolloClient.mutation(LoginWithEmailMutation(email,
   password)).execute()

val token = response.data?.loginWithEmail

securePreferences.saveSessionToken(token)
```

Die `SecurePreferences` Klasse speichert den JWT Token in den EncryptedSharedPreferences. SharedPreferences ermöglichen es, Daten in einer Key-Value Form zu spei-

chern. EncryptedSharedPreferences verschlüsseln die Daten, bevor sie gespeichert werden. Bei zukünftigen Anfragen wird der JWT Token aus den EncryptedSharedPreferences geladen und über einen `HTTP-Interceptor` als Authorization Header in den Anfragen an den Server gesetzt.

```
fun provideApolloClient(
    securePreferences: SecurePreferences
): ApolloClient {
    val okHttpClient = OkHttpClient.Builder()
        .addInterceptor { chain ->
            val original = chain.request()
            val builder = original.newBuilder().method(original.
               method, original.body)
            builder.header("Authorization", "Bearer␣${
               securePreferences.getSessionToken()}")
            chain.proceed(builder.build())
        }
        .build()

    return ApolloClient.Builder()
        .serverUrl("https://api.linkwave.org/graphql/")
        .okHttpClient(okHttpClient)
        .build()
}
```

5.4.4 Unter Windows

Die Benutzerverwaltung auf Windows wird durch die Windows-App verwaltet. Hier können sich Benutzer:innen registrieren, einloggen und ihr Profil bearbeiten. Es ist möglich sich mit einer Email Adresse und einem Passwort oder mit einem Google Konto anzumelden. Dafür wird die GraphQL API des LinkWave Servers verwendet. Die Anfragen werden über die `GraphQLHttpClient` Klasse von `GraphQL.Client` gesendet. Es werden GraphQL Queries und Mutations zu erstellen und zu senden verwendet. Um Benutzer:innen anzumelden wird eine `loginWithEmail` Anfrage geschickt oder eine `LoginOAuthGoogleAccessToken` Anfrage, wenn sich Benutzer:innen mit Google anmelden. Wenn sich Benutzer:innen erfolgreich angemeldet haben, wird der JWT Token in den Local Settings gespeichert.

Registrierungsprozess

Beim Registrierungsprozess geben Benutzer:innen ihre persönlichen Daten ein, darunter Benutzername, Vor- und Nachname, E-Mail-Adresse und Passwort. Diese Daten werden mittels eines GraphQL-Requests an den Server gesendet. Nach erfolgreicher Registrierung initiiert das System automatisch den Prozess zur Generierung eines SSL-Zertifikats für das Gerät des:der Benutzer:in. Dieses Zertifikat wird verwendet, um die Identität des Geräts innerhalb des Netzwerks zu authentifizieren und eine verschlüsselte Verbindung zu anderen Geräten herzustellen.
Für die Registrierung wird die Mutation `Register` verwendet, die die Benutzerdaten an den Server sendet und den Registrierungsprozess startet. Nachdem Benutzer:innen erfolgreich registriert wurde, wird ein JWT-Token zurückgegeben, das zur Authentifizierung von Benutzer:innen bei zukünftigen Anfragen verwendet wird, und das JWT-Token wird in den “Local Settings” gespeichert.

```
public async Task RegisterAsync(User user)
{
    string username = UsernameTextBox.Text;
    string email = EmailTextBox.Text;
    // Weitere Benutzerdaten
    string password = PasswordBox.Password;
    ...
    var signUpRequest = new GraphQLRequest
    {
        Query = //GraphQL Query um Benutzer zu registrieren
        Variables = // Variablen fuer die Query
    };
    var signUpResponse = await graphQLClient.SendQueryAsync<
       SignUpResponseData>(signUpRequest);
    deviceCertificate = await Task.Run(async () => await
       Certificate.CompleteCertificateSigningProcess());
    // Weitre Schritte
    App.GetService<ILocalSettingsService>().SaveSettingAsync("
       JWTToken", loginResponse.Data.Token)
}
```

Anmeldeprozess

Die Benutzer:innen können sich mit ihrer Email Adresse und ihrem Passwort oder mit ihrem Google Konto über Google OAuth anmelden und authentifizieren. Beim Anmeldeprozess geben Benutzer:innen ihre Anmeldeinformationen ein, die an den Server gesendet werden, um die Identität von Benutzer:innen zu überprüfen. Wenn die Anmeldeinformationen korrekt sind, wird ein JWT-Token zurückgegeben, das zur Authentifizierung der Benutzer:innen bei zukünftigen Anfragen verwendet wird. Nach einem erfolgreichen Login wird das JWT-Token in den “Local Settings” gespeichert.
Wenn der User breits ein Konto hat aber sich zum ersten mal auf einem Gerät anmeldet, wird sein Gerät automatisch registriert und ein SSL-Zertifikat für das Gerät generiert.
Für die Anmeldung des Users wird `LoginWithEmail` Mutation verwendet.

```
private async void LoginButton_Click(object sender, Microsoft.UI.
    Xaml.RoutedEventArgs e)
{
    var loginRequest = new GraphQLRequest
    {
        Query = //GraphQL Query um Benutzer zu anmelden,
        Variables = // Variablen fuer die Query
    };
    var loginResponse = await graphQLClient.SendQueryAsync<
        LoginResponse>(loginRequest);
}
```

Für die Anmeldung des Users wird Google OAuth verwendet. Dies ist viel komplexer und aufgrund der vielen Schritte werden hier nicht alle Methoden aufgeführt.

```
string authorizationRequest = // Authorization Request
// Starten des Browser mit dem Authorization Request
private async Task<string?> performCodeExchangeAsync(string code,
    string code_verifier)
{
    var loginRequest = new GraphQLRequest
    {
        Query = //GraphQL Query um Benutzer zu anmelden,
        Variables = // Variablen fuer die Query
    };
    GraphQLResponse<LoginResponse> loginResponse;
    loginResponse = await graphQLClient.SendQueryAsync<
        LoginResponse>(loginRequest);
    return loginResponse.Data.Token;
}
// Weitere Schritte
```

5.4.5 Im Web

Das Web-Dashboard emöglicht den Benutzer:innen ihr Konto zu verwalten. Hier kann registriert, eingeloggt und das Profil bearbeitet werden. Es ist möglich sich mit einer Email Adresse und einem Passwort oder mit einem Google Konto anzumelden. Dafür wird die GraphQL API des LinkWave Servers verwendet. Für die Kommunikation mit der GraphQL API wird die Bibliothek `graphql-request` [18] verwendet. Das Dashboard ist in SvelteKit geschrieben. Da SvelteKit ein Server-Side-Rendering Framework ist, wird die GraphQL API direkt vom Server aufgerufen. Der SvelteKit Server fungiert als ein Proxy für die GraphQL API.

Email und Passwort

Um Benutzer:innen anzumelden wird eine `loginWithEmail` Anfrage an den Server geschickt. Diese Anfrage enthält die Email und das Passwort von Benutzer:innen. Die Email und das Passwort werden über Form Actions an den SvelteKit server gesendet. Der Server sendet die Anfrage an den LinkWave Server und erhält den JWT Token zurück. Dieser wird in einem Cookie gespeichert, um ihn für zukünftige Anfragen zu verwenden.

```
const loginRes = await gqlClient.request(loginWithEmailMutation, {
    email: form.data.email,
    password: form.data.password
});
const sessionToken = loginRes.loginWithEmail;

event.cookies.set('sessionToken', sessionToken, {
    httpOnly: true,
    maxAge: 60 * 60 * 24 * 7,
    path: '/',
    sameSite: 'lax',
    secure: !dev
});
```

- **httpOnly**: Der Cookie kann nicht über Client-Side JavaScript zugegriffen werden. Das erhöht die Sicherheit, da der JWT Token nicht durch XSS Angriffe gestohlen werden kann.
- **maxAge**: Der Cookie läuft nach einer Woche ab. Die Benutzer:innen müssen sich nach einer Woche erneut anmelden.
- **path**: Der Cookie ist auf der gesamten Webseite verfügbar.
- **secure**: Der Cookie wird nur über HTTPS gesendet, wenn die Webseite im Produktionsmodus ist.

Google Konto

Die Anmeldung mit einem Google Konto (OAuth) funktioniert auf dem Web-Dashboard über die `google-auth-library` Bibliothek. Die Benutzer:innen werden auf die Google OAuth Seite weitergeleitet, wo sie sich mit ihrem Google Konto anmelden können. Nachdem die Benutzer:innen sich angemeldet haben, werden sie auf eine Callback-URL weitergeleitet, die den authorization code als Query Parameter enthält. Dieser authorization code wird an die LinkWave GraphQL API gesendet.

```
google: async () => {
    const redirectUri = googleOAuthClient.generateAuthUrl({
        accessType: 'offline',
        scope: [
            'https://www.googleapis.com/auth/userinfo.email',
            'https://www.googleapis.com/auth/userinfo.profile'
        ]
    });

    return redirect(302, redirectUri);
}
```

Die Callback URL wird auf dem SvelteKit Server definiert. SvelteKit bietet die Möglichkeit, sogenannte Server-Routen zu definieren. Das sind einfache Rest Endpoints. Nachdem die Benutzer:innen sich mit ihrem Google Konto angemeldet haben, werden

sie auf so eine Server-Route weitergeleitet. Diese Route liest den authorization code aus den Query Parametern und sendet ihn an die LinkWave GraphQL API.

```
export const GET: RequestHandler = async (event) => {
    const code = event.url.searchParams.get('code');
    const loginRes = await gqlClient.request(
       loginWithGoogleDocument, {
        authorizationCode: code
    });
    const sessionToken = loginRes.loginOAuthGoogle;
    setSessionCookie(event, sessionToken);
    return redirect(302, '/');
};
```

5.4.6 Am Server

JWT Implementierung

LinkWave hat die Authentifizierung über JSON Web Tokens (JWT) implementiert, die auf der .NET-Plattform aufbaut. Die Implementierung nutzt Kernklassen aus dem System.IdentityModel.Tokens.Jwt Paket und unterstützende Sicherheitsklassen.

JwtProvider Klasse Die `JwtProvider` Klasse ist verantwortlich für die Erzeugung von JWTs und Refresh Tokens. Sie ist so konzipiert, dass sie die Einstellungen aus der `JwtSettings` Konfiguration erhält und diese anwendet, um die Token zu generieren.

```
private JwtSettings _settings;
public JwtProvider(IOptions<JwtSettings> settings)
{
    _settings = settings.Value;
}
public string GenerateToken(User user)
{
    var claims = new[]
    {
        new Claim("UserId", user.Id.ToString()),
        new Claim(ClaimTypes.Email, user.Email),
        ...
    };
    var token = new JwtSecurityToken(
        issuer: _settings.Issuer,
        audience: _settings.Audience,
        ...
    );
    var tokenValue = new JwtSecurityTokenHandler().WriteToken(token
       );
    return tokenValue;
}
```

OAuth Implementierung

LinkWave verwendet die Google OAuth API für Authentifizierungsdienste, um eine sichere Überprüfung der Benutzeridentitäten und den sicheren Abruf ihrer Informationen zu gewährleisten.

Token-Verifikation und Benutzerdatenabruf

Die `GoogleOAuth`-Klasse kapselt die Logik für die Interaktion mit Google's OAuth 2.0 Endpoint. Der Hauptprozess beinhaltet:

1. Die Erzeugung eines `RestClient`, der sich an `"https://oauth2.googleapis.com"` richtet.
2. Die Konfiguration eines `RestRequest`, um den Token per POST-Methode zu beantragen.
3. Die Verwendung von Anwendungsdaten wie `redirectUri`, `clientId` und `clientSecret`, die basierend auf dem `clientType` dynamisch aus den Einstellungen geladen werden.
4. Die Rückgabe einer Antwort vom Google-Server als String.

Hier ist ein Ausschnitt des Codes, der die Verifikation eines Google-Tokens darstellt:

```
public async Task<string?> VerifyGoogleTokenAsync(...)
{
    ...
    RestClient client =
    new RestClient("https://oauth2.googleapis.com");
    RestClient client =
    new RestClient("https://oauth2.googleapis.com");
    RestRequest request = new RestRequest("token", Method.Post);
    ...
    return response.Content;
}
```

Anpassungen für Mobile OAuth-Authentifizierung

Die Methode LoginOAuthGoogleAccessToken wurde extra implementiert, um den OAuth-Prozess auf mobilen Geräten zu unterstützen, da diese Geräte nur mit Access Tokens arbeiten und keinen Authorization Code zurückliefern.

Zusammenfassung der Anmeldeprozesse

Zwei Hauptmethoden `LoginOAuthGoogle` und `LoginOAuthGoogleAccessToken` ermöglichen es Benutzer:innen, sich über Google OAuth anzumelden. Der Standardprozess verwendet den Authorization Code, während die alternative Methode direkt mit dem Access Token arbeitet.

```
public async Task<string?> LoginOAuthGoogle(...)
{
    ...
    GoogleTokenResponse accessToken = ...
    Database.Models.User? userData =
    _googleOAuth.GetUserData(accessToken.AccessToken);
    ...
}

public async Task<string?> LoginOAuthGoogleAccessToken(...)
{
    ...
    Database.Models.User userData =
    _googleOAuth!.GetUserData(accessToken);
    ...
}
```

In beiden Fällen werden die Benutzerdaten aus den Google-Kontoinformationen extrahiert und in der Datenbank gespeichert. Anschließend wird ein JWT für Benutzer:innen generiert und zurückgegeben, mit dem sich Benutzer:innen authentifizieren können.

Two Factor Authentication Implementierung

Die technische Umsetzung auf dem Server erfolgt über zwei Hauptfunktionen:

```
public async Task<string> GenerateQrCodeUri(...)
{
    var twoFactorAuthenticator = new TwoFactorAuthenticator();
    var secretKey = Utilities.GenerateRandomString(32).Result;
    ...
    // Generierung des QR-Codes
    ...
    return setupInfo.QrCodeSetupImageUrl;
}

public async Task<bool> ValidateTwoFactorAuthentication(...)
{
    ...
    // Validierung des TOTP-Codes
    ...
    return twoFactorAuthenticator.ValidateTwoFactorPIN(secretKey,
       token);
}
```

5.5 Hintergrundausführung

Die Hintergrundausführung ist eine sehr wichtige aber für den User nicht sichtbare Funktion. Sie ermöglicht es, dass die App auch im Hintergrund weiterläuft und Daten empfangen und senden kann. LinkWave soll sich für einen User wie ein Bestandteil des Betriebssystems anfühlen.

Anfragen für Dateiübertragungen sollen auch ohne geöffnete App durchgeführt werden können. Wenn ein User eine Datei senden möchte, soll er dies auch tun können, wenn die App geschlossen ist. Außerdem soll die Zwischenablage ohne jegliche Interaktion mit der App übetragen werden. Im Idealfall sollte ein User nicht einmal merken, dass die App im Hintergrund läuft. Die App wird daher eigentlich nur für die Konfiguration benötigt.

Trotzdem soll ein User wissen, dass die App im Hintergrund läuft. Es soll auch möglich sein, die App leicht komplett zu schließen oder die Einstellungen aufzurufen. Rin User soll die App wie eine Erweiterung des Betriebssystems benutzen können. Damit soll eine Erfahrung geschaffen werden, die der eines Ökosystems ähnlich ist.

5.5.1 Unter iOS

Die Implementation der Hintergrundausführung auf iOS ist in der jetzigen Version von LinkWave nicht implementiert. Dies liegt daran, dass Apps auf iOS Geräten in einer Sandbox laufen. Apple ermöglicht es Entwickler:innen nicht, diese Sandbox durchgehend im Hintergrund laufen zu lassen.

Apple möchte, dass Apps das Background Update Paradigma verwenden. Dies bedeutet, dass Apps eine Update Funktion definieren, die von iOS in einem regelmäßigen Intervall aufgerufen wird. Dieses Intervall kann von iOS dynamisch angepasst werden. In dieser Funktion sollen Apps Daten aktualisieren und Benachrichtigungen versenden. Danach wird die App wieder in den Schlafmodus versetzt. (siehe Apple Developer Documentation [5])

Da LinkWave durchgehend dafür ausgelegt ist, Daten zu übertragen, würde das Versetzen in den Schlafmodus die Funktionalität der App stark einschränken. Daher ist es nicht sinnvoll, die Hintergrundausführung auf iOS zu implementieren. Benutzer:innen können jedoch weiterhin alle Funktionen der App nutzen, wenn sie die App geöffnet haben.

5.5.2 Unter macOS

Im Gegensatz zu iOS ist es auf macOS möglich, Hintergrundprozesse zu implementieren. Dies ist sogar mit einer Sandbox möglich. Hierfür muss man einfach nur dafür sorgen, dass ein UI Element duchgehend vorhanden ist. Als UI Element kann ein Menüleisten Icon verwendet werden. (Siehe Kapitel 3.2) Dieses Icon ermöglicht es zudem, über ein Kontextmenü die App zu schließen oder die Einstellungen zu öffnen. Ein Menüleisten Icon lässt sich ganz einfach mit dem `MenuBarExtra` Element in einer SwiftUI View erstellen.

```
MenuBarExtra {
  MenuBarExtraContent() // Inhalt des Kontextmenues
} label: {
  Image("menubar-icon") // Menueleisten Icon Bild
}
```

5.5.3 Unter Android

Unter Android werden langlaufende Hintergrundprozesse durch sogenannte `Services` realisiert. Es gibt zwei Arten von Services: Foreground Services und Background Services. Seit Android 8.0 (API Level 26) gibt es Einschränkungen für Background Services, um die Akkulaufzeit zu verbessern. Diese Einschränkungen haben Background Services faktisch obsolet gemacht. Stattdessen sollten Foreground Services verwendet werden, die eine Benachrichtigung anzeigen und somit dem Benutzer signalisieren, dass ein Service im Hintergrund ausgeführt wird. Der `LinkWaveService` ist ein `Foreground Service` und beinhaltet die Funktionen, die im Hintergrund ausgeführt werden sollen. Das Beinhaltet das Empfangen von Dateien, das Empfangen der geteilten Zwischenablage und das Bewerben des Geräts im Netzwerk über Bonjour. Nachstähende Abbildung zeigt das Klassendiagramm des `LinkWaveService`.

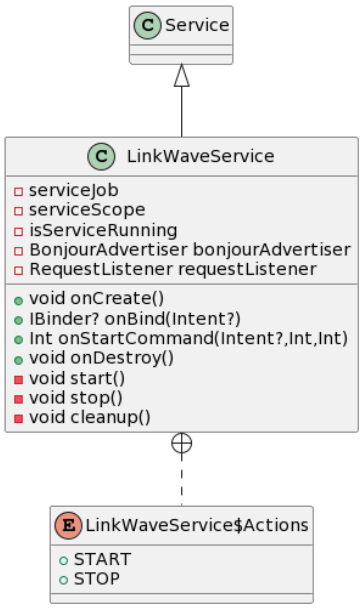


Abbildung 55: LinkWaveService Klassendiagramm

Die `onStartCommand()` Funktion wird aufgerufen, wenn der Service einen Intent bekommt. Dieser Intent kann entweder eine `START`- oder `STOP`-Action sein. Wie die Namen schon vermuten lassen, wird der Service mit einer `START`-Action gestartet und mit einer `STOP`-Action gestoppt.

start() In der `start()` Funktion wird der `BonjourAdvertiser` gestartet, der das Gerät im Netzwerk bewirbt und der `RequestListener`, der auf Anfragen von anderen Geräten wartet. Außerdem wird die Benachrichtigung erstellt, die den Benutzer:innen signalisiert, dass der Service im Hintergrund läuft.

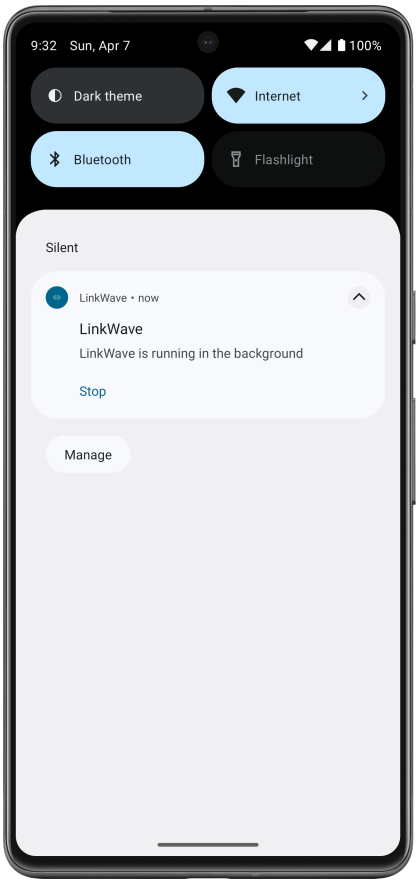


Abbildung 56: LinkWaveService Benachrichtigung

stop() In der `stop()` Funktion wird der `BonjourAdvertiser` und der `RequestListener` gestoppt. Alle Coroutines, die im Service laufen, werden abgebrochen und die Benachrichtigung wird entfernt.

5.5.4 Unter Windows

Ähnlich wie unter macOS ist es auch unter Windows möglich, mit einem Tray Icon Hintergrundprozesse zu implementieren. Dieses Icon wird in der Taskleiste angezeigt und ermöglicht es, die App zu schließen oder die Einstellungen zu öffnen. Das Tray Icon wird mit der `NotifyIcon` Klasse erstellt. Diese Klasse ermöglicht es, ein Icon in der Taskleiste anzuzeigen und darauf zu reagieren.

```
var trayIcon = new NotifyIcon();
trayIcon.Text = "LinkWave";
trayIcon.Icon = new System.Drawing.Icon(Path.Combine(
    AppContext.BaseDirectory, "Assets/TrayIcon.ico"));
trayIcon.Visible = true;
```

Um ein Kontextmenü zu erstellen, wird die `ContextMenuStrip` Klasse verwendet. Diese wird dem `NotifyIcon` zugewiesen und wird angezeigt, wenn Benutzer:innen mit Rechtsclick auf das Icon klicken.

```
var trayMenu = new ContextMenuStrip();
trayMenu.Items.Add("Open", null, OnOpenWindow);
trayMenu.Items.Add("Open␣Settings", null, OnOpenSettings);
trayMenu.Items.Add("Send␣File", null, OnOpenSendFile);
trayMenu.Items.Add("Quit", null, OnQuit);
trayIcon.ContextMenuStrip = trayMenu;
```

Durch das Überschrieben der Closed Methode wird gewährleistet, dass das Schließen des Hauptfensters nicht automatisch die App beendet. In dieser Methode wird das Hauptfenster nur versteckt, anstatt es zu schließen. Zusätzlich wird dem Betriebssystem mitgeteilt, dass die Operation erfolgreich war, damit die App nicht vom Betriebssystem beendet wird.

```
private void MainWindow_Closed(object sender, WindowEventArgs e)
{
    e.Handled = true;
    this.Hide();
}
```

5.6 Verschlüsselung

Die Verschlüsselung ist ein sehr wichtiger Bestandteil von LinkWave. Da LinkWave auch in unsicheren Netzwerken, wie beispeislweise gratis WLAN in Cafés, verwendet werden soll, ist es wichtig, dass die Daten verschlüsselt übertragen werden. Es soll nicht möglich sein, dass ein Angreifer über LinkWave gesendete Dateien abfangen kann. Zudem soll auch nicht erkennbar sein, wann Dateien und welche übertragen werden.

Trotzdem sollen die Daten sehr schnell übertragen werden. Es soll nicht zu einer spürbaren Verzögerung kommen. Die Verschlüsselung soll also sehr effizient sein. Zudem soll die Verschlüsselung auf einem oft verwendeten Standard basieren, damit die sichere und vor allem idente Implementierung auf allen Plattformen möglich ist.

LinkWave verwendet für jegliche Verbindungen TCP. TCP kümmert sich jedoch nicht um die Verschlüsselung der Daten. Daher muss LinkWave selbst für die Verschlüsselung sorgen.

5.6.1 Funktionsweise

LinkWave verfügt über eine eigene 'Certificate Authority', die zum Signieren von Zertifikaten verwendet wird. Diese Zertifikate werden verwendet, um die Kommunikation zwischen den Geräten zu verschlüsseln. Dieser Prozess ist in der folgenden Abbildung dargestellt. Es ist wichtig zu erwähnen, dass die Schritte eins und zwei nur bei der ersten Anmeldung oder Registrierung auf einem Gerät durchgeführt werden. Die Zertifikate werden dann auf dem Gerät an einem sicheren Ort gespeichert, z. B. im Certificate Store unter Windows oder in der Keychain unter macOS. Diese werden dann verwendet, um die Kommunikation zwischen den Geräten zu verschlüsseln.

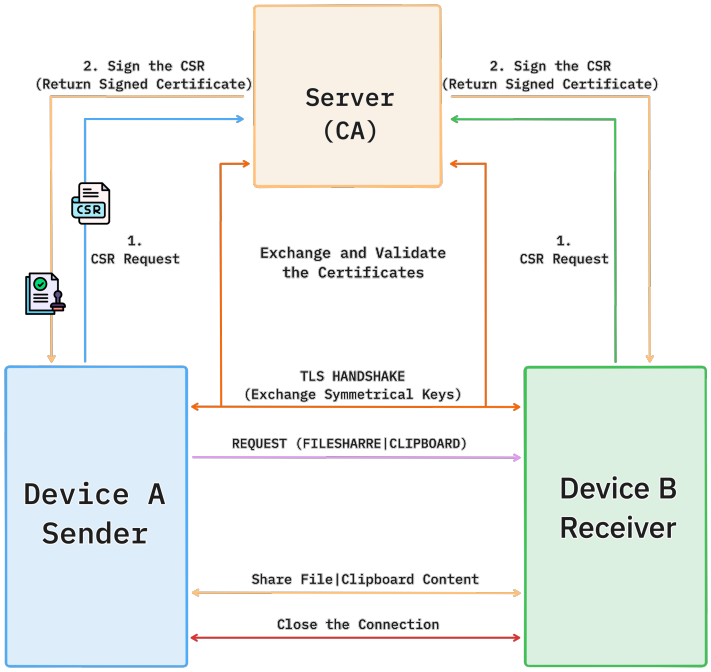


Abbildung 57: SSL/TLS

1. **CSR Erstellung:** Device A und B generieren ein Certificate Signing Request (CSR) und senden es an den Server.
2. **CSR Signierung:** Der Server signiert die CSR und stellt ein SSL-Zertifikat aus, das an Device A und B zurückgesendet wird.
3. **TLS-Handshake:** Mit den erhaltenen Zertifikaten führen Device A und Device B

einen TLS-Handshake durch. Dabei werden die Zertifikate ausgetauscht und validiert.

4. **Symmetrischer Schlüsselaustausch:** Im Zuge des Handshakes wird ein symmetrischer Verschlüsselungsschlüssel generiert.

5. **Beginn der verschlüsselten Kommunikation:** Alle nachfolgenden Daten werden mit dem symmetrischen Schlüssel verschlüsselt übertragen.

6. **Trennung der Verbindung:** Nach der Datenübertragung wird die Verbindung sicher getrennt, während die Sicherheit der übertragenen Daten erhalten bleibt.

Was ist SSL/TLS?

Über SSL und TLS SSL und TLS sind kryptografische Protokolle, die entwickelt wurden, um eine sichere Datenübertragung über unsichere Netzwerke wie das Internet zu ermöglichen. Ursprünglich von Netscape entwickelt, wurde SSL schnell zum Standard für sichere Webkommunikation, bevor es durch TLS, seine verbesserte und sicherere Nachfolgeversion, ersetzt wurde. Beide Protokolle bieten Mechanismen für die Verschlüsselung und Authentifizierung von Daten, die zwischen Webbrowsern und Webservern ausgetauscht werden, und finden darüber hinaus auch in anderen Protokollen wie E-Mail, VoIP und Instant Messaging Anwendung.

Der SSL/TLS-Handshake Der Aufbau einer SSL/TLS-gesicherten Verbindung ist ein mehrstufiger Prozess, bekannt als der "Handshake". Dieser Prozess legt die Grundlage für eine sichere Kommunikation, indem er die Identität der Kommunikationspartner überprüft und einen gemeinsamen Verschlüsselungsschlüssel etabliert.

Vereinbarung Im ersten Schritt des Handshakes vereinbaren Client und Server, welche Protokollversion verwendet wird und wählen Algorithmen für die Verschlüsselung und Authentifizierung. Diese Auswahl stellt sicher, dass beide Parteien die gleichen kryptografischen Standards verwenden, was für die Sicherheit der Datenübertragung essenziell ist.

Schlüsselaustausch Nach der Vereinbarung der zu verwendenden Protokolle und Algorithmen tauschen die Kommunikationspartner Schlüsselinformationen aus. Dieser Schritt umfasst oft die Authentifizierung des Servers (und manchmal auch des Clients) mittels Zertifikaten. Die ausgestellten Zertifikate, signiert von einer vertrauenswürdigen Zertifizierungsstelle (CA), ermöglichen es den Parteien, die Identität des Gegenübers zu verifizieren und einen gemeinsamen symmetrischen Verschlüsselungsschlüssel zu generieren.

Verschlüsselte Kommunikation Mit dem erfolgreichen Abschluss der Authentifizierung und des Schlüsselaustausches beginnt die verschlüsselte Kommunikation über den gesicherten Kanal. Alle übertragenen Daten werden mit dem vereinbarten symmetrischen Schlüssel verschlüsselt, was eine vertrauliche und integre Datenübertragung gewährleistet.

Die Bedeutung von Zertifikaten Zertifikate spielen eine zentrale Rolle im SSL/TLS-Handshake. Sie enthalten den öffentlichen Schlüssel des Zertifikatinhabers sowie Identitätsinformationen und sind von einer Zertifizierungsstelle signiert. Diese Zertifikate sind das Mittel zur Verifizierung der Identität des Servers und zur Verhinderung von Man-in-the-Middle-Angriffen, bei denen Angreifer versuchen, die Kommunikation abzufangen.

Quelle: [8]

Implementierung

Im Folgenden wird der Verschlüsselungsprozess im Detail erläutert:

5.6.2 Unter iOS und macOS

Die Verschlüsselung auf iOS und macOS wird durch die Verwendung von SSL/TLS-Zertifikaten erreicht. Diese kann man einfach in einem `NWParams` Objekt an eine `NWConnection` übergeben. Dieses Objekt wird dann verwendet, um eine Verbindung zu einem Server aufzubauen. Die Verbindung wird dann automatisch verschlüsselt.

Im ersten Schritt muss man ein Public-Private Key Pair generieren. Dies kann mit der `SecKeyCreateRandomKey` Funktion erreicht werden. Diese Funktion speichert es zudem in der Keychain.

```
let tag = tlsKeysTag.data(using: .utf8)!
let attributes: [String: Any] = [
    kSecAttrKeyType as String: kSecAttrKeyTypeRSA,
    kSecAttrKeySizeInBits as String: 2048,
    kSecPrivateKeyAttrs as String: [
        kSecAttrIsPermanent as String: true,
        kSecAttrApplicationTag as String: tag,],]
var error: Unmanaged<CFError>?
guard let privateKey = SecKeyCreateRandomKey(attributes as
   CFDictionary, &error) else { return }
let publicKey = SecKeyCopyPublicKey(privateKey)
```

Im zweiten Schritt muss man ein Zertifikat generieren, welches der Server signiert. Dieses Zertifikat wird dann für die Verschlüsselung verwendet. Eine Signierungsanfrage nennt sich CSR (Certificate Signing Request). Dieses CSR wird mithilfe der Bibliothek `CertificateSigningRequest` erstellt. Dieses CSR wird dann an den Server gesendet, welcher es signiert und zurücksendet. Hierfür wird die `signCertificate` Mutation benutzt.

```
let csr = CertificateSigningRequest(
  commonName: commonName,
  organizationName: organizationName,
  organizationUnitName: organizationName, countryName: countryName,
  keyAlgorithm: algorithm
)
let publicKeyBits =
  SecKeyCopyExternalRepresentation(publicKey, nil)! as Data
let builtCSR = csr.buildCSRAndReturnString(
  publicKeyBits, privateKey: privateKey, publicKey: publicKey)
var query = Objects.Mutation.signCertificate(csrPem: builtCSR)
```

Dieses Zertifikat wird in der Keychain gespeichert, um es später für die Verschlüsselung zu verwenden. Dies kann mit der `SecItemAdd` Funktion erreicht werden. Danach muss man das Zertifikat nur noch mit einem Label versehen, um es später wiederzufinden.

```
guard let derData = Data(base64Encoded: pemBody) else {
    return false }
guard let certificate = SecCertificateCreateWithData(nil, derData
   as CFData) else {
    return false }
let addquery: [String: Any] = [
    kSecClass as String: kSecClassCertificate,
    kSecValueRef as String: certificate]
var status = SecItemAdd(addquery as CFDictionary, nil)
if status == errSecSuccess { } else { return }
status = SecCertificateSetPreferred(
    certificate, TLSCertLabel as CFString, nil)
```

Um eine verschlüsselte Verbindung aufzubauen, muss man zuerst ein `TLSOptions` Objekt erstellen. Dieses Objekt enthält die Zertifikate, die für die Verschlüsselung verwendet werden. Zudem legt es die Überprüfung der Zertifikate anderer Gesprächsparteien fest.

```
let identity = retrieveIdentityFromKeychain()
let tlsOptions = NWProtocolTLS.Options()
guard let os_sec_identy = sec_identity_create(identity) else
{ return }
sec_protocol_options_set_local_identity(
    tlsOptions.securityProtocolOptions, os_sec_identy)
sec_protocol_options_set_min_tls_protocol_version(
    tlsOptions.securityProtocolOptions, .TLSv12)
sec_protocol_options_set_verify_block(
    tlsOptions.securityProtocolOptions,
{ sec_protocol_metadata, sec_trust, sec_protocol_verify_complete in
  // Zertifikat ueberpruefen
}, DispatchQueue.main)
```

In der Funktion zum Überprüfen der Zertifikate wird das Zertifikat an den Server gesendet. Dieser sendet dann eine Antwort, ob das Zertifikat gültig ist. Wenn das Zertifikat gültig ist, wird die Verbindung aufgebaut. Wenn nicht, wird die Verbindung abgebrochen. Dies wird über die `isCertificateValid` Query erreicht.

```
let secTrust = sec_trust_copy_ref(sec_trust).takeRetainedValue()
let serverCertificates = SecTrustCopyCertificateChain(secTrust)
let serverCertificate = serverCertificates[0] as SecCertificate
let pemCert = convertToPEMCertificate(serverCertificate)
let query = Objects.Query.isCertificateValid(certPem: pemCert)
```

Mitfile dieser `TLSOptions` kann nun eine `NWConnection` erstellt werden. Diese Verbindung ist dann verschlüsselt und kann für die Datenübertragung verwendet werden.

```
let tlsOptions = getTLSOptions()
let params = NWParameters(tls: tlsOptions, tcp: .init())
let connection = NWConnection(to: endpoint, using: params)
```

5.6.3 Unter Android

Die Verschlüsselung auf Android wird durch die Verwendung von SSL/TLS-Zertifikaten erreicht. Diese Zertifikate werden verwendet, um die Kommunikation zwischen den Geräten zu verschlüsseln. Der Prozess beginnt mit der Generierung eines Schlüsselpaares und der Erstellung eines CSR (Certificate Signing Request). Dieser CSR wird mithilfe der `SignCertificate` Mutation an den Server gesendet, der ihn signiert und ein SSL-Zertifikat ausstellt. Dieses Zertifikat wird dann auf dem Gerät gespeichert und für die Verschlüsselung der Kommunikation verwendet.

```
mutation SignCertificate($csrPem: String!) {
    signCertificate(csrPem: $csrPem)
}
```

Schlüsselerzeugung

Das `SSLHelper` Objekt stellt die Methoden zur Schlüsselerzeugung und Zertifikatsanfrage bereit. Die Methode `generateAndStoreKeyPair()` erzeugt ein Schlüsselpaar und speichert es in der Android KeyStore. Die Methode `generatePemCsr` erstellt eine Zertifikatsanfrage und gibt sie als PEM-String zurück.

```
val keyPair = generateAndStoreKeyPair("linkwave_keypair")
val subject =
    X500Name("CN=${LocalDeviceInfo.name},␣O=linkwave.org")
val csrBuilder = JcaPKCS10CertificationRequestBuilder(
    subject,
    keyPair.public
)
val signer = JcaContentSignerBuilder("SHA256withRSA").build(keyPair
   .private)
val csr = csrBuilder.build(signer)

return csrToPem(csr)
```

Zertifikatsignierung

Für die Signierung der Zertifikatsanfrage wird die `SignCertificate` Mutation an den Server gesendet. Der Server signiert die Anfrage und gibt das signierte Zertifikat als PEM-String zurück.

```
val response = apolloClient.mutation(SignCertificateMutation(csr))
    .execute()
val signedCertificate = response.data?.signCertificate ?: throw
   Exception("No signed certificate found")
SSLHelper.saveSignedCertificateToKeystore(signedCertificate)
```

Das signierte Zertifikat wird im Android KeyStore gespeichert und kann für die Verschlüsselung der Kommunikation verwendet werden.

Verschlüsselte Kommunikation

Die verschlüsselte Kommunikation wird durch die Verwendung des `SSLContext` Objekts erreicht. Dieses Objekt wird mit dem signierten Zertifikat initialisiert und kann dann für die Erstellung von `SSLSocket` Objekten verwendet werden. Folgender Code zeigt die Erstellung eines `SSLContext` Objekts mit dem signierten Zertifikat.

```
val privateKeyEntry = SSLHelper.getPrivateKeyEntry()
val keyManagerFactory = KeyManagerFactory.getInstance(
   KeyManagerFactory.getDefaultAlgorithm()).apply {
    init(KeyStore.getInstance(KeyStore.getDefaultType()).apply {
        load(null)
        setKeyEntry("alias", privateKeyEntry.privateKey, null,
           privateKeyEntry.certificateChain)
    }, null)
}

val sslContext = SSLContext.getInstance("TLS").apply {
    init(keyManagerFactory.keyManagers, null, null)
}
```

Jetzt kann das `SSLContext` Objekt verwendet werden, um eine SSLFactory zu erstellen, die dann für die Erstellung von `SSLSocket` Objekten verwendet werden kann.

```
val sslSocketFactory = sslContext.socketFactory
val sslSocket = sslSocketFactory.createSocket() as SSLSocket
```

Dieser `SSLSocket` kann dann für die sichere Kommunikation mit anderen Geräten verwendet werden.

```
val outputStream = sslSocket.outputStream
val inputStream = sslSocket.inputStream

outputStream.write("Hello, World!".toByteArray())
val response = inputStream.read()
```

5.6.4 Unter Windows

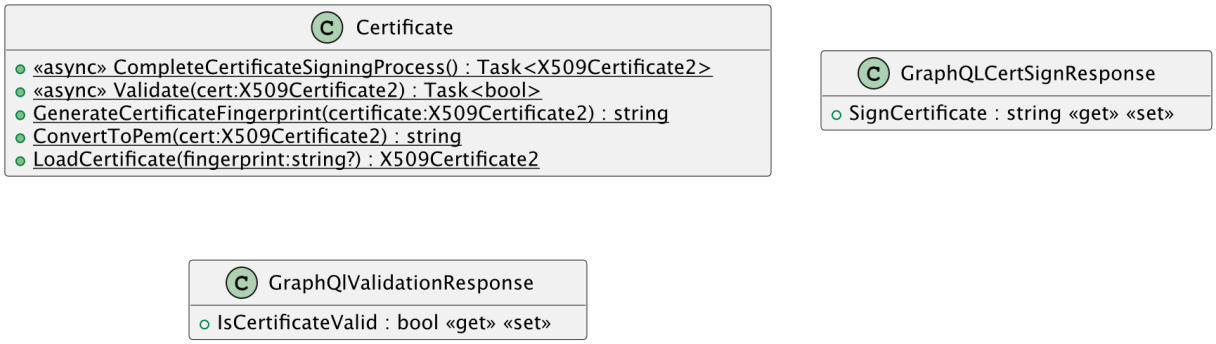


Abbildung 58: UML Certificate

Certificate Klasse

Die Certificate Klasse in LinkWave umfasst die Methoden zur Generierung von Zertifikatsanfragen (CSR), der Kommunikation mit dem GraphQL-API zur Signierung der Anfragen und der Verifizierung von Zertifikaten.
Die CompleteCertificateSigningProcess Methode koordiniert die Erstellung und Signierung eines Zertifikats, während die Validate Methode das empfangene Zertifikat überprüft.

```
public static async Task<X509Certificate2>
   CompleteCertificateSigningProcess()
{
    var cert = GenerateCertificateSigningRequest();
    var signedCertPem = await SendCsrForSigning(cert.csr);
    var signedCertBytes = PemToDer(signedCertPem);
    var importedCert =
    ImportSignedCertificate(signedCertBytes, cert.privateKey,
       GenRandomPassword());
    return importedCert;
}

public static async Task<bool> Validate(X509Certificate2 cert)
```

```
{
    var certPem = ConvertToPem(cert);
    return await ValidateCertificateByServerAsync(certPem);
}
```

1. **Certificate Signing Request (CSR)**: Dieser Prozess beschreibt die Anforderung eines Zertifikats durch ein Gerät. Dabei wird eine Zertifikatsanfrage an die Zertifizierungsstelle Certificate Authority (CA) gerichtet. Nach der Überprüfung signiert die CA die Anfrage und gibt ein Zertifikat aus, welches das Gerät zur Authentifizierung seiner Identität nutzt. [11]

2. **X509Certificate2**: Diese Klasse gehört zur .NET-Bibliothek und dient der Verwaltung von X.509-Zertifikaten. Ihre Funktionen umfassen das Importieren, Exportieren und Validieren von Zertifikaten. [12]

SSL Klasse

Die SSL-Klasse ist für die Etablierung und Sicherstellung von SSL/TLS-Verbindungen zuständig. Sie beinhaltet Methoden für den Server und den Client, um SSL-Handshakes durchzuführen und so eine sichere Verbindung herzustellen.

```
public static async Task<SslStream>
   AcceptAndAuthenticateClientAsync(...)
{
    TcpClient client = await listener.AcceptTcpClientAsync();
    SslStream sslStream = new SslStream( client.GetStream(),
    leaveInnerStreamOpen: false,
    new RemoteCertificateValidationCallback(ValidateCertificate)
       ,...);
    await sslStream.AuthenticateAsServerAsync(serverCertificate,
       ...);
    return sslStream;
}
```

5.6.5 Am Server

GraphQL Endpoints

CSR Signierung

Auf dem Server enthält die Mutation CA die Methode SignCertificate, die eine einem Gerät zugeordnete CSR entgegennimmt und den signierten Zertifikatsstring zurückgibt. Die eigentliche Signatur wird von der Methode SignCsr der Klasse CAServer ausgeführt.

```
public async Task<string> SignCertificate(..., string csrPem)
{
    var device = await dbContext.Devices.
    FirstOrDefaultAsync(x => x.Id == deviceId);
    if (device == null) return null;
    string signedCert = LinkWaveServices.
    Utilities.Security.CAServer.SignCsr(csrPem);
    return signedCert;
}


public static string SignCsr(string csrPem)
{
    Pkcs10CertificationRequest csr = ParseCsr(csrPem);
    X509V3CertificateGenerator certGen = new
       X509V3CertificateGenerator();
    certGen.SetPublicKey(csr.GetPublicKey());
    certGen.AddExtension(X509Extensions.AuthorityKeyIdentifier,
       ...);
    certGen.AddExtension(X509Extensions.BasicConstraints, ...);
    X509Certificate signedCert = certGen.Generate(...);
    return ConvertToPem(signedCert);
}
```

Certificate Validierung

Auf der Serverseite wird die Methode ValidateCertificate verwendet, um die Gültigkeit eines Zertifikats zu überprüfen.
Die Methode ValidateCertificate nimmt ein Zertifikat als PEM-String entgegen und gibt einen boolschen Wert zurück, der angibt, ob das Zertifikat gültig ist.

```
public bool IsCertificateValid(string certPem)
{
    return LinkWaveServices.Utilities.
    Security.CA.ValidateCertificate(certPem);
}
```

6 Zusammenfassung

Mit AirDrop hat Apple die Interoperabilität von Geräten revolutioniert. Apple-Geräte tauschen im Hintergrund ständig Daten aus. Wird auf einem Gerät etwas in die Zwischenablage kopiert, kann es auf einem anderen Gerät “wie von Zauberhand” wieder eingefügt werden. Die Vielfalt der Geräte und Betriebssysteme sowie das monopolistische Verhalten von Apple erschweren die Interoperabilität zwischen Apple-Geräten und Geräten anderer Hersteller erheblich.

LinkWave bringt diese Funktionen in Form von nativen Apps auf die vier großen Plattformen, Windows, MacOS, Android und iOS. Ähnlich wie mit AirDrop können Dateien, Texte und Bilder nahtlos zwischen verschiedenen Geräten ausgetauscht werden. Durch die lokale Funktionsweise von LinkWave wird der Datenaustausch deutlich schneller und sicherer gestaltet, als dies mit bereits bestehenden Cloud-Alternativen möglich ist.

Zentral ist die plattformübergreifende Architektur, die eine konsistente und intuitive Benutzererfahrung ermöglicht und gleichzeitig hohe Sicherheitsstandards durch End-to-End-Verschlüsselung gewährleistet. Die Anwendung nutzt modernste Technologien wie SwiftUI und Jetpack Compose, um die Integration und Funktionalität über Plattformgrenzen hinweg zu optimieren.

Ausblickend könnte LinkWave weiterentwickelt werden, um noch mehr Gerätetypen und Plattformen zu unterstützen, was die universelle Zugänglichkeit und Nützlichkeit der App weiter steigern würde. Zukünftige Weiterentwicklungen könnten sich darauf konzentrieren, die Effizienz der Datenübertragung weiter zu erhöhen und die Anwendung um zusätzliche Funktionalitäten zu erweitern, die den Bedürfnissen der Benutzer:innen noch besser entsprechen.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] WinUI Microsoft Dokumentation https://learn.microsoft.com/de-de/windows/apps/winui/

[2] Swift GraphQ https://swift-graphql.com/introduction

[3] KeychainSwift https://github.com/evgenyneu/keychain-swift

[4] Jetpack Compose https://developer.android.com/develop/ui/compose

[5] Apple Background Updates https://developer.apple.com/documentation/uikit/app_and_environment/scenes/preparing_your_ui_to_run_in_the_background/using_background_tasks_to_update_your_app

[6] Kotlin Flow https://kotlinlang.org/docs/flow.html

[7] Kotlin Serialization https://kotlinlang.org/docs/serialization.html

[8] TLS/SSL https://www.cloudflare.com/de-de/learning/ssl/transport-layer-security-tls/

[9] Chilli Cream https://chillicream.com/docs/hotchocolate/v13

[10] Microsoft SSLStream Dokumentation https://learn.microsoft.com/en-us/dotnet/api/system.net.security.sslstream?view=net-8.0

[11] Microsoft CSR Dokumentation https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.x509certificates.certificaterequest.createsigningrequest?view=net-8.0

[12] Microsoft X509Certificate2 Dokumentation https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.x509certificates.x509certificate2?view=net-8.0

[13] Google OAuth https://developers.google.com/identity/protocols/oauth2

[14] JWT https://jwt.io/introduction

[15] Two Factor Authentication https://learn.microsoft.com/de-de/aspnet/core/security/authentication/mfa?view=aspnetcore-8.0

[16] HOTP vs TOTP https://rublon.com/blog/hotp-totp-difference/

[17] Das Bonjour Protokoll https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/NetServices/Introduction.html

[18] GraphQL Request https://www.npmjs.com/package/graphql-request