Flurfunk – ein LoRa-basierter   
Hausflohmarkt

Ein Bild, das Text, Grafiken, Schrift, Logo enthält.

KI-generierte Inhalte können fehlerhaft sein.

Bericht zum Projekt für «Distributed Programming   
and Internet Architecture»

Universität Basel, Sommersemester 2025

vorgelegt von Andrea Seehuber, 05.07.2025

Inhaltsverzeichnis

[1 Einleitung 1](#_Toc202648225)

[1.1 Motivation 1](#_Toc202648226)

[1.2 Zielsetzung 1](#_Toc202648227)

[2 Umsetzung des Projekts 1](#_Toc202648228)

[2.1 Rahmenbedingungen 1](#_Toc202648229)

[2.2 Implementierung 2](#_Toc202648230)

[2.2.1 Hardwarenahe Kommunikation 2](#_Toc202648231)

[2.2.2 Nachrichtenrouting und Netzwerkprotokoll 3](#_Toc202648232)

[2.2.3 Synchronisationslogik 3](#_Toc202648233)

[2.2.4 Konsistenzmodell und CRDT-Kompatibilität 4](#_Toc202648234)

[2.2.5 Verschlüsselung und Datenschutz 6](#_Toc202648235)

[3 Test und Evaluation 6](#_Toc202648236)

[4 Ausblick 7](#_Toc202648237)

[5 Literaturverzeichnis 8](#_Toc202648238)

# Einleitung

## Motivation

Digitale Kommunikation ist heutzutage fast immer auf zentrale Infrastrukturen angewiesen: Server, Mobilfunkmasten, Internetprovider. In Krisensituationen, abgelegenen Regionen oder autoritär regierten Staaten kann diese Abhängigkeit jedoch problematisch werden. Zentrale Server lassen sich zensieren, kontrollieren oder schlicht abschalten. Die permanente Internetverbindung ist zudem energieintensiv und schliesst Menschen ohne Zugang aktiv aus. Dezentrale Netzwerke bieten hier eine alternative Kommunikationsform: Sie funktionieren ohne zentrale Steuerung, sind robuster gegen Ausfälle und ermöglichen einen direkten, selbstbestimmten Austausch von Informationen. Projekte wie «Meshtastic» zeigen, wie durch den Einsatz von LoRa-Funkmodulen einfache Peer-to-Peer-Netzwerke aufgebaut werden können, die Nachrichten energieeffizient über mehrere hundert Meter oder sogar Kilometer übertragen – ganz ohne Mobilfunk oder WLAN (Meshtastic LLC, unbekannt).

LoRa (Long Rang) ist ein proprietäres Modulationsverfahren auf Basis von Chirp Spread (CSS), das auf lizenzfreien ISM-Bändern (z. B. 868 MHz in Europa) arbeitet. Es ermöglicht extrem stromsparende Kommunikation bei sehr geringer Bandbreite, typischerweise im Bereich von wenigen hundert Bit pro Sekunde. Dadurch eignet es sich ideal für batteriebetriebene Geräte und einfache Datenübertragung über grosse Entfernungen bei gleichzeitig hoher Störfestigkeit (Union schweizerischer Kurzwellen-Amateure, 2024).

## Zielsetzung

Das Projekt «Flurfunk» setzt genau auf diese Idee: Eine App, die Bewohner eines Hauses über ein LoRa-basiertes Mesh-Netzwerk miteinander verbindet, um gebrauchte Gegenstände auszutauschen. Dabei erfolgt sämtliche Datenübertragung ohne Anbindung an das Internet und lokal. Beim ersten Start der App gibt der Benutzer seine Adressdaten ein, die zur eindeutigen Identifizierung im Netzwerk dienen. Dann erlaubt die Benutzeroberfläche die Erstellung neuer Angebote in verschiedenen Kategorien. Die eigenen Angebote können deaktiviert werden, falls sich eine andere Person dafür interessiert und auch wieder reaktiviert werden. Im deaktivierten Zustand werden sie auf dem GUI der Netz-Peers nicht mehr angezeigt. Angebote können vom eigenen GUI auch komplett ausgeblendet werden, wenn sie physisch nicht mehr vorhanden sind. Alle Daten werden periodisch und verschlüsselt innerhalb des eigenen Netzwerks per Broadcast übertragen.

# Umsetzung des Projekts

## Rahmenbedingungen

Für die Kommunikation über das LoRa-Mesh kamen zwei Waveshare SX1261 LoRa USB-Adapter (850 – 920 MHz) zum Einsatz (Digitec Galaxus AG, 2025). Diese Geräte bieten eine einfache Möglichkeit, LoRa-Kommunikation per USB-UART-Schnittstelle mit einem beliebigen Host-System aufzubauen. Sie unterstützen einen Stream- und einen Packet-Modus und lassen sich über   
AT-Befehle steuern. Um eine zuverlässige Kommunikation im europäischen ISM-Band zu gewährleisten, wurde auf den Geräten vor Inbetriebnahme nach Anleitung auf der Website des Herstellers ein Firmware-Update durchgeführt (Waveshare, unbekannt). Die Konfiguration erfolgte zunächst über die Android-App «Serial USB Terminal» und ein Austausch von Nachrichten konnte erfolgreich getestet werden.

Die beiden USB-Dongles wurden jeweils über einen USB-OTG-Adapter direkt mit zwei Android-Smartphones verbunden:

* Sony Xperia 5 II (Android 12)
* Samsung Galaxy A16 (Android 15)

Beide Geräte unterstützen USB-OTG und ermöglichen so eine direkte serielle Kommunikation zwischen App und LoRa-Dongle. Der Zugriff auf die USB-Schnittstelle erfolgt über das Android-USB-Host-API in Verbindung mit einer eigenen USB-Abstraktionsschicht innerhalb der App.

Die App wurde in Java mit Android Studio entwickelt. Dabei kamen die offiziellen Android-Entwicklungswerkzeuge sowie das USB-Host-API und AndroidX-Bibliotheken zum Einsatz. Für die serielle Kommunikation mit dem LoRa-Dongle wurde die Open-Source-Bibliothek «usb-serial-for-android» verwendet (Kai Morich, 2025). Die App ist modular aufgebaut und unterscheidet klar zwischen der hardwarenahen Steuerung des LoRa-Dongles, dem Netzwerkprotokoll und der verteilten Datenhaltung.

## Implementierung

### Hardwarenahe Kommunikation

Die App kommuniziert direkt mit dem angeschlossenen LoRa-Dongle über die serielle USB-Schnittstelle. Die Umsetzung orientiert sich dabei an der Funktionsweise der Android-App «USB Serial Terminal» und ist zweistufig aufgebaut (Kai Morich, 2024):

* Die Klasse LoRaService ist ein persistenter Android-Service, der von der Netzwerkkomponente ATLoRaManager gebunden werden kann. Er kapselt die gesamte Lebensdauer der Verbindung zum Dongle und delegiert die eigentliche I/O-Kommunikation an ein internes LoRaSocket-Objekt. Die Methoden connect, write und disconnect dienen als API für die Anwendung.
* Die Klasse LoRaSocket verwaltet die konkrete serielle Verbindung. Sie öffnet den USB-Port, konfiguriert ihn mit Standardparametern und startet dann einen Hintergrundthread über die Klasse SerialInputOutputManager, der kontinuierlich auf eingehende Daten wartet. Diese werden bei Empfang an eine LoRaListener-Instanz weitergegeben, die als Callback-Schnittstelle dient. Falls der Lese-Thread unerwartet abbricht, versucht LoRaSocket bis zu drei Wiederverbindungen, bevor die Verbindung endgültig aufgegeben wird. Dadurch bleibt die App auch bei instabiler Verbindung funktionsfähig.

Der verwendete LoRa-Dongle übernimmt intern die Fragmentierung und Pufferung von Datenpaketen. Nutzdaten bis 960 Byte werden automatisch in Pakete à 240 Byte aufgeteilt und eine zuverlässige Übertragung der Datenpakete wird durch eine integrierte CRC-Prüfung sichergestellt. Fehlerhafte Pakete werden vom Dongle automatisch verworfen (Waveshare, unbekannt).

### Nachrichtenrouting und Netzwerkprotokoll

Die Klasse ATLoRaManager bildet die Schnittstelle zwischen der LoRa-Hardware und dem Anwendungscode. Nach dem Start übernimmt sie die Initialisierung des Dongles per   
AT-Befehlen (z. B. AT+MODE=1 für den Stream-Modus) und setzt alle relevanten Übertragungsparameter. Da serielle Daten über USB nicht paketorientiert, sondern in beliebig grossen Chunks eintreffen, ist ein robuster Parser erforderlich. Hierfür wird ein interner SerialFrameReader eingesetzt, der alle eingehenden Bytes puffert und nach einem definierten Nachrichtenrahmen durchsucht: Gemäss Netzwerkprotokoll beginnt eine Nachricht mit einem # und endet mit der Zeichenkette ;EOM. Um sicherzustellen, dass der Empfangspuffer korrekt mit Chunks arbeitet und keine Überläufe oder Fehlinterpretationen auftreten, wird die Parserlogik thread-sicher umgesetzt. Für den ausgehenden Nachrichtenfluss wird eine Sende-Queue verwendet, die in einem dedizierten Thread verarbeitet wird. Um Kollisionen zu vermeiden, wird vor jedem Sendevorgang überprüft, ob aktuell ein Empfang stattfindet, und ein Mindestabstand zwischen zwei Übertragungen erzwungen.

Vollständig rekonstruierte Nachrichten werden über eine Callback-Schnittstellt im Stil eines Consumer<String> an die registrierte Anwendungsschicht weitergereicht. In der MainActivity der App wird hierfür der MessageDispatcher registriert. Der Dispatcher übernimmt daraufhin folgende Aufgaben:

* **Protokoll-Parsing.** Die Nachricht wird mithilfe der Klasse Protocol analysiert und in ihre Bestandteile zerlegt.
* **Mesh-ID-Filterung.** Nur Nachrichten mit passender MID (Mesh ID) werden weiterverarbeitet. Die MID wird beim Erstellen des Benutzerprofils aus der eingegebenen Adresse erzeugt.
* **Duplikatprüfung.** Bereits verarbeitete Nachrichten-IDs werden zwischengespeichert und ignoriert, um redundante Verarbeitung zu verhindern.
* **Befehlsbasiertes Routing.** Je nach Protokollkomando (SYNOF, REQOF, PRDAT, etc.) wird die Nachricht an eine zuständige Komponente delegiert:
  + OfferSyncManager für Angebotsdaten und
  + PeerSyncManager für Benutzerprofile.

Dieses Routingmodell ist modular aufgebaut und erlaubt eine einfache Erweiterung des Protokolls um neue Nachrichtentypen.

### Synchronisationslogik

Die App basiert auf einem vollständig dezentralen, rein broadcast-basierten Synchronisationsmechanismus. Um den Zustand aller Angebote und Benutzerprofile innerhalb eines gemeinsamen LoRa-Meshs abzugleichen, kommen zwei Synchronisationsprozesse zum Einsatz, die in der MainActivity alle 90 Sekunden und mit einem zeitlich versetzten Abstand von 5 Sekunden aufgerufen werden:

* Peer-Synchronisation zum Abgleich und Import von Benutzerprofilen – SYNPR
* Angebots-Synchronisation zum Abgleich und Import von Flohmarktangeboten – SYNOF

Die Klasse PeerSyncManager verwaltet alle Vorgänge zum Austausch von Benutzerprofilen und basiert auf drei Protokollnachrichten:

* SYNPR enthält eine Liste aller bekannten Peer-IDs samt Änderungszeitpunkten.
* REQPR fordert anhand dieser Liste veraltete oder fehlende Profile gezielt an.
* PRDAT überträgt vollständige Profilinformationen verschlüsselt.

Die Klasse OffferSyncManager ist für den Austausch und Import von Angebotsdaten zuständig. Sie arbeitet nach einem ähnlichen Mechanismus:

* SYNOF enthält eine Liste der bekannten Angebote samt Änderungszeitpunkten.
* REQOF fordert anhand dieser Liste gezielt veraltete oder fehlende Angebote an.
* OFDAT überträgt die vollständigen Angebotsdaten verschlüsselt.

Alle empfangenen Daten werden lokal gespeichert, mit bestehenden Einträgen abgeglichen und aktualisiert, wenn der übertragene Zeitstempel höher ist als der vorhandene.

Sowohl Benutzer- als auch Angebotsdaten werden als JSONObjects verschickt. Eine Angebots-Objekt hat z. B. folgendes Format:

{"category":"ELECTRONICS",

"createdAt":1751717068181,

"creatorId":"b40d774ae4dea2df",

"deleted":false,

"description":"Samsung Galaxy A16, nicht 5G fähig",

"lastModified":1751717068181,

"offerId":"c0e4d418134fca5c",

"status":"ACTIVE",

"title":"Smartphone"}

Jedem Attribut des Objekts ist ein eindeutiger Protokollschlüssel zugeordnet, mit dessen Hilfe die Informationen für die Darstellung auf dem GUI extrahiert werden können.

Die maximale Grösse einer LoRa-Nachricht ist durch die Hardware auf 960 Bytes limitiert. Um grössere Datenmengen trotzdem senden zu können, werden sowohl Peer- als auch Angebotsdaten beim Versand in mehrere Blöcke (Chunks) aufgeteilt. Dies geschieht explizit im Code durch manuelle Aufteilung der Daten in JSONArray-Fragmente.

### Konsistenzmodell und CRDT-Kompatibilität

Die Synchronisation der Daten erfolgt in der App nach dem Prinzip eines asynchronen, eventual consistent Systems, bei dem jeder Peer jederzeit aktualisierte Informationen senden oder empfangen kann. Dieses Modell orientiert sich an der Idee von CRDTs, wobei in diesem Projekt zwei relevante CRDT-Typen zur Anwendung kommen (Shapiro, Preguiça, Baquero, & Zawirski, 2011):

* **Das Benutzerprofil wird als Last-Write-Wins Register (LWW) angelegt.** Jedem Benutzer wird einmal eine eindeutig identifizierbare UID zugewiesen sowie ein Zeitstempel TS, der bei Änderungen am Profil aktualisiert wird. Dabei gilt: Wenn ein empfangenes Profil einen neueren TS hat als das lokal gespeicherte, werden nur die betroffenen Felder überschrieben.
* **Angebote werden als LWW-Element-Set angelegt.** Jedes neu erstellte Angebot erhält wiederum eine eindeutig identifizierbare OID sowie einen TS, der bei Änderung des Angebots aktualisiert wird. Der Status eines Angebots kann entweder «aktiv» oder «inaktiv» sein, Angebotsdetails können nicht verändert werden.

Jeder Benutzer hält neben seinem eigenen Profil noch ein LWW-Element-Set mit allen Profilen, inklusive seinem eigenen, welches per Broadcast periodisch synchronisiert wird. Einmal synchronisierte Profile bzw. Angebote werden nie gelöscht, können aber auf «inaktiv» gestellt werden, um sie vom GUI auszublenden. Dabei können Angebote vom Besitzer selbst inaktiviert werden oder durch ein automatisches Timeout (3 Wochen). Löscht ein Benutzer sein Profil, wird einmalig eine letzte Broadcast-Nachricht versendet, mit der allen anderen Peers im Netz signalisiert wird, dass die Angebote dieses Benutzers inaktiviert werden können. Zusätzlich gibt es für jeden Benutzer einen lastSeen-Zeitstempel LS, der bei jeder Aktivität aktualisiert wird. Wenn LS durch Abgleich mit der lokalen Zeit einen bestimmten Grenzwert überschritten hat, werden ebenfalls alle Angebote des fraglichen Benutzers inaktiviert.

Damit bleibt die Systemkonsistenz erhalten: Die zugrundeliegenden CRDT-Strukturen bleiben vollständig, während einzelne Daten im GUI situationsabhängig gefiltert werden. Nachfolgend ist der Algorithmus zur Aktualisierung der lokalen Angebotsliste dargestellt:

public static void updateOrAdd(List<Offer> offers, Offer newOffer) {  
 for (int i = 0; i < offers.size(); i++) {  
 Offer existing = offers.get(i);  
 if (existing.getOfferId().equals(newOffer.getOfferId())) {  
 existing.merge(newOffer);  
 return;  
 }  
 }  
 offers.add(newOffer);  
}

public void merge(Offer other) {  
 if (!this.offerId.equals(other.offerId)) return;  
 if (other.lastModified > this.lastModified) {  
 this.title = other.title;  
 this.description = other.description;  
 this.category = other.category;  
 this.status = other.status;  
 this.deleted = other.deleted;  
 this.lastModified = other.lastModified;  
 }  
}

private void updateTimestamp() {  
 this.lastModified = System.*currentTimeMillis*();  
}

Da der TS im updateTimestamp-Aufruf lokal über System.currentTimeMillis gesetzt wird, basiert die Konfliktauflösung implizit auf der Systemzeit. Dies kann in seltenen Fällen zu Inkonsistenzen führen, wenn Geräte abweichende Uhren verwenden.

### Verschlüsselung und Datenschutz

Um die Vertraulichkeit der übertragenen Daten zu gewährleisten, verschlüsselt die App alle Nachrichteninhalte mit Klartext vor dem Senden mit dem symmetrischen Verschlüsselungsverfahren AES im CTR-Modus. Der Schlüssel wird deterministisch aus der Mesh-ID der jeweiligen Hausgemeinschaft abgeleitet. Dazu wird die MID mittels SHA-256 gehasht und die ersten   
16 Bytes als AES-Schlüssel verwendet. Die MID wird aus den eingegebenen Adressdaten berechnet, indem alle Leer- und Sonderzeichen entfernt, der Text in Kleinbuchstaben umgewandelt und anschliessend per SHA-256 gehasht wird. Aus dem Hash wird ein kompaktes, 6 Byte langes Präfix extrahiert und Base64-kodiert, um eine kurze, aber stabile MID zu erzeugen. Auf diese Weise besitzen alle Benutzerprofile mit derselben Adresse denselben Schlüssel, ohne dass dieser explizit gespeichert werden muss. Jede Nachricht wird zusätzlich mit einem zufällig generierten Initialisierungsvektor IV versehen, der gemeinsam mit dem verschlüsselten Text im Netzwerk übertragen wird. Die Kombination aus deterministischem Schlüssel und zufälligem IV garantiert, dass identische Nachrichten bei jeder Übertragung zu unterschiedlichen Chiffren führen. Die Entschlüsselung erfolgt analog, indem der Schlüssel erneut aus der MID abgeleitet und der Klartext aus dem verschlüsselten Inhalt der Nachricht und dem IV rekonstruiert wird.

# Test und Evaluation

Die App wurde während der Entwicklung kontinuierlich getestet – primär direkt auf den Android-Smartphones unter realen Bedingungen mit verbundenem LoRa-Dongle. Die Überprüfung erfolgte sowohl visuell über die grafische Oberfläche als auch durch gezielte Analyse der Logcat-Ausgabe. Für die beiden Netzwerkklassen PeerSyncManager und OfferSyncManger wurden automatisierte Unit-Tests implementiert, um die korrekte Verarbeitung und CRDT-konforme Konfliktauflösung der synchronisierten Daten zu prüfen.

Ein wiederkehrendes Problem im Testprozess war der Umgang mit den USB-Datenpaketen, die vom Dongle in fragmentierter Form übertragen werden. Die Daten mussten korrekt zusammengesetzt werden, bevor daraus vollständige JSONObjects rekonstruiert werden konnten. Besonders bei grossen Nachrichten (z. B. Listen mit vollständigen Angebotsdaten) kam es wiederholt zu korrupten Datenchunks, weil der Dongle zu früh wieder in den Sendemodus wechselte, bevor ein vollständiger Empfang abgeschlossen war. Dies führte zu JSON-Parsing-Fehlern. Um mit diesen Fehlern umzugehen, wurde in der Angebots-Synchronisation handleOfferData ein Robustheitsmechanismus implementiert, der unvollständige oder fehlerhafte JSON-Elemente extrahiert und ignoriert, damit zumindest die korrekten Angebote verarbeitet werden können. In anderen Modulen wurde auf solche Mechanismen verzichtet, was weiterhin zu Problemen führen kann.

Insgesamt zeigt die App trotz dieser Herausforderungen ein stabiles Verhalten in typischen Nutzungsszenarien. Die Synchronisation erfolgt zuverlässig, solange gültige Nachrichten empfangen werden können, und alle Benutzeraktionen auf dem GUI werden korrekt im Mesh verarbeitet. Screenshots der graphischen Benutzeroberfläche sowie von relevanten Logging-Ereignissen finden sich im Anhang des Berichts.

# Ausblick

Mit der Flurfunk-App konnte ein funktionierender Prototyp für ein vollständig dezentrales Kommunikationssystem auf Basis von LoRa implementiert werden. Im Fokus standen dabei Robustheit, Energieeffizienz und ein CRDT-kompatibles Datenmodell zur Sicherstellung der Systemkonsistenz. Für den produktiven Einsatz ergeben sich jedoch verschiedene Erweiterungs- und Verbesserungsmöglichkeiten:

* **Nachrichtenweiterleitung.** Ein ursprünglich implementierter Mechanismus zur paketbasierten Weiterleitung über mehrere Peers (sogenanntes «Hopping» wie z. B. in Meshtastic realisiert) wurde wieder entfernt, da sich damit in der Praxis keine stabile Zustellung realisieren liess. Zukünftig könnte eine robustere Hopping-Logik mit TTL-Feldern, Duplikaterkennung und explizitem Routing wieder aufgenommen werden.
* **Resilienz gegenüber fragmentierten und fehlerhaften Nachrichten.** Die systematische Behandlung beschädigter oder unvollständiger Datenpakete ist bisher nur teilweise implementiert. Ein generisches, fehlertolerantes Paket-Parsing wäre hier sinnvoll.
* **Sicherheit und Datenschutz.** Die aktuelle symmetrische Verschlüsselung auf Basis der Mesh-ID ist pragmatisch, aber unsicher gegen gezielte Angriffe. Eine Erweiterung um asymmetrische Schlüsselpaare und Signaturen wäre sinnvoll, um Manipulation und Identitätsdiebstahl zu verhindern.
* **GUI- und UX-Verbesserungen.** Viele Funktionen (z. B. Inaktivität von Angeboten oder Peers) werden nur logisch verarbeitet, aber nicht explizit im Frontend dargestellt. Eine bessere visuelle Rückmeldung über den Zustand des Netzwerks würde die Transparenz und Nutzbarkeit verbessern.

Trotz aller Limitierungen zeigt die App exemplarisch, wie sich mit einfachen Mitteln ein robustes, selbstorganisiertes Kommunikationssystem aufbauen lässt. Der gesamte Code steht auf GitHub zur Verfügung: <https://github.com/Antimon80/Flurfunk>

# Literaturverzeichnis

Digitec Galaxus AG. (2025). *Digitec - Development Boards and Kits*. Abgerufen am 04. Juli 2025 von Digitec: https://www.digitec.ch/en/s1/product/waveshare-sx1262-lora-tcxo-usb-adapter-850-930mhz-development-boards-kits-53127368

Kai Morich. (2024). *GitHub Repository*. Abgerufen am 04. Juli 2025 von SimpleUsbTerminal: https://github.com/kai-morich/SimpleUsbTerminal

Kai Morich. (2025). *usb-serial-for-android*. Abgerufen am 04. Juli 2025 von GitHub Repository: https://github.com/mik3y/usb-serial-for-android

Meshtastic LLC. (unbekannt). *Meshtastic*. Abgerufen am 04. Juli 2025 von Meshtastic: https://meshtastic.org/

Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011). *A comprehensive studiy of Convergent and Commutative Replicated Data Types.* Paris: Inria - Centre Paris Rocquencourt.

Union schweizerischer Kurzwellen-Amateure. (2024). *Was ist LoRa?* Abgerufen am 04. Juli 2025 von USKA: https://uska.ch/lora/

Waveshare. (unbekannt). *USB-TO-LoRa-xF*. Abgerufen am 04. Juli 2025 von Waveshare Wiki.

# Redlichkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte Hilfe angefertigt habe. Es wurden keine weiteren Quellen als jene, die im Literaturverzeichnis ausgewiesen sind, verwendet.

Ich habe im Rahmen der Erstellung dieser Arbeit die Sprach-KI ChatGPT (OpenAI) genutzt, um mir bei der Formulierung von Textabschnitten, der Überprüfung technischer Konzepte sowie bei der Erklärung der verwendeten Datenmodelle und Algorithmen helfen zu lassen. Alle durch ChatGPT generierten Inhalte wurden kritisch geprüft, technisch überarbeitet und in eigener Verantwortung in den Gesamtkontext eingeordnet. Das Logo der App ist KI-generiert.

Basel, 05.07.2025



# Anhang

Graphische Benutzeroberfläche

|  |  |  |
| --- | --- | --- |
| Ein Bild, das Text, Screenshot, Brief, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Startbildschirm beim  erstmaligen Öffnen der App | Erstellung des Benutzerprofils | Menüleiste mit  Angebotskategorien |
| Ein Bild, das Text, Elektronik, Screenshot, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Screenshot, Software, Webseite enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Screenshot, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Erstellung eines Angebots | Listenansicht einer  Angebotskategorie | Detailansicht eines  fremden Angebots |
| Ein Bild, das Text, Screenshot, Webseite, Software enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Elektronik, Screenshot, Schrift enthält.  KI-generierte Inhalte können fehlerhaft sein. | Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Listenansicht der eigenen  Angebote (farbig hinterlegt je nach Aktivitätsstatus) | Detailansicht eines  eigenen Angebots | Einstellungen zum Ändern der eigenen Kontaktdaten |
| Ein Bild, das Text, Screenshot, Schrift, Design enthält.  KI-generierte Inhalte können fehlerhaft sein. |  |  |
| Impressum |  |  |

Relevante Logcat-Ereignisse

|  |
| --- |
| Ein Bild, das Screenshot, Muster, Stoff, monochrom enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Die maximale Grösse einer vollständig rekonstruierbaren Protokollnachricht liegt gemäss Hersteller bei 960 Bytes, was durch systematisches Testen verifiziert werden konnte. |
| Ein Bild, das Text, Screenshot, Schrift, Schwarzweiß enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Eine Broadcastnachricht mit allen beim Peer lokal vorhandenen Angebots-IDs und den zugehörigen Zeitstempeln wurde empfangen und für das Protokollparsing rekonstruiert. |
| Ein Bild, das Text, Screenshot, Schwarzweiß, Schwarz enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Datenpakete, die eine Grösse von 960 Byte überschreiten, werden in kleinere Pakete aufgeteilt und jeweils als eigenständige Protokollnachrichten unmittelbar hintereinander versendet. |
|  |
| Eine Liste mit verschlüsselten Angebotsinformationen wurde empfagen, erfolgreich geparst und lokal gespeichert. |
| Ein Bild, das Text, Screenshot, Schwarzweiß, monochrom enthält.  KI-generierte Inhalte können fehlerhaft sein. |
| Timing-Probleme führen gelegentlich dazu, dass falsch decodierte Nachrichten nicht geparst werden können. |