FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Interdisziplinäres Projekt
Radio Adblocker

Inhaltsverzeichnis

Inhaltsverzeichnis

Umfang des Projekts

Frontend

Ausführliche Dokumentation

Backend

Erweiterte Inhalte in Abschlussdokument

- Ausführlicher als in der Präsentation
- Weitere Gedanken von uns niedergeschrieben
- Recherche-Ergebnisse als Grundlage für Weiterentwicklung
- “Getting Started”-Dokumente zum Aufsetzen und Erklärung des Source-Codes

Inhaltsverzeichnis

Aufgabe

Eine Radio-App, ohne Werbeunterbrechung

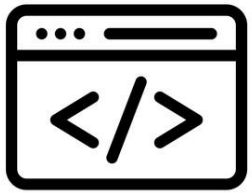
Anforderungen

- Radio-App
- Konfiguration der Vorlieben
- Werbung blockieren

Anforderungen

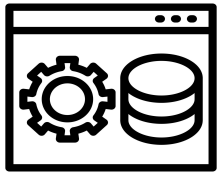
- **Radio-App**
- Konfiguration der Vorlieben
- Werbung blockieren

Teams



Frontend

Ahmad Fadi Aljabri
Nicolas Harrje
Nils Stegemann



Backend

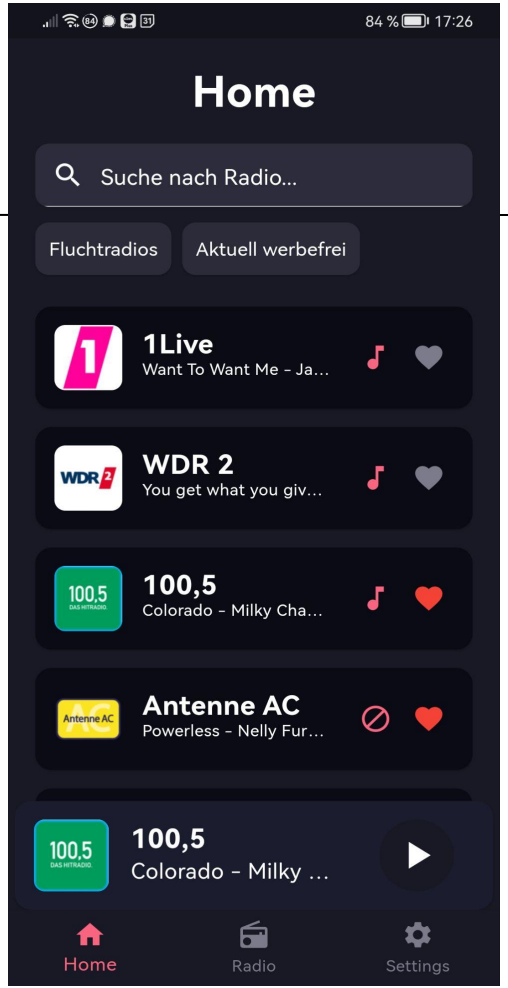
Chris Henkes
Kaan Yazici
Maximilian Breuer
Gerrit Weiermann

Radio-App

- Entwicklung mit Flutter
- Aufteilung auf drei Screens
  - Homescreen
  - Radioscreen
  - Settings

Radio-App Homescreen

- Anzeige aller verfügbaren Radios mit
  - Logo
  - Songname + Interpret
  - Indikator (Werbung / keine Werbung)
  - Favoritenstatus
- Anzeige des aktuell gespielten Radios

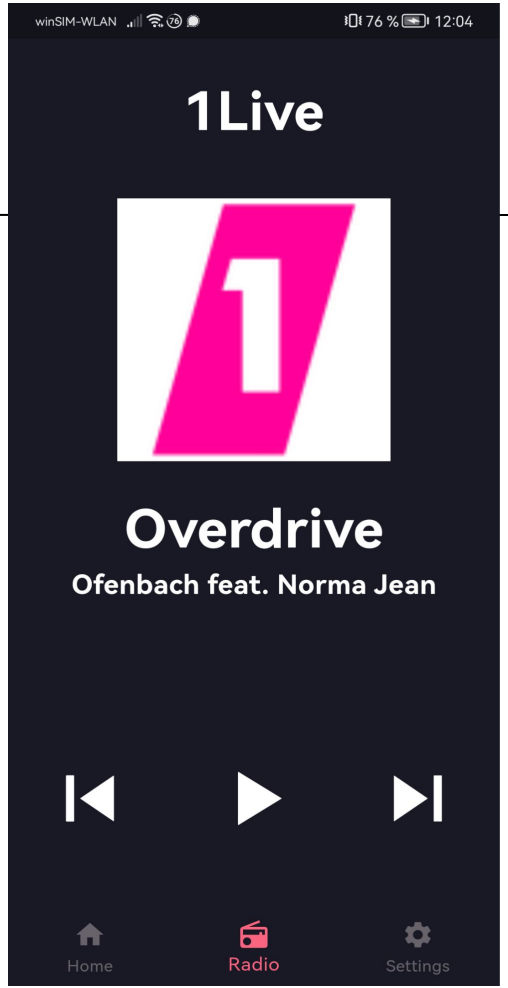


Radio-App Homescreen

- Filterung der Radios
  - textuelle Suche
  - Fluchtradios
  - Aktuell werbefrei

Radio-App Radioscreen

- Fokus auf aktuelles Radio
- Anzeigen von Metadaten
- Vor- und zurückschalten durch Radio-Liste

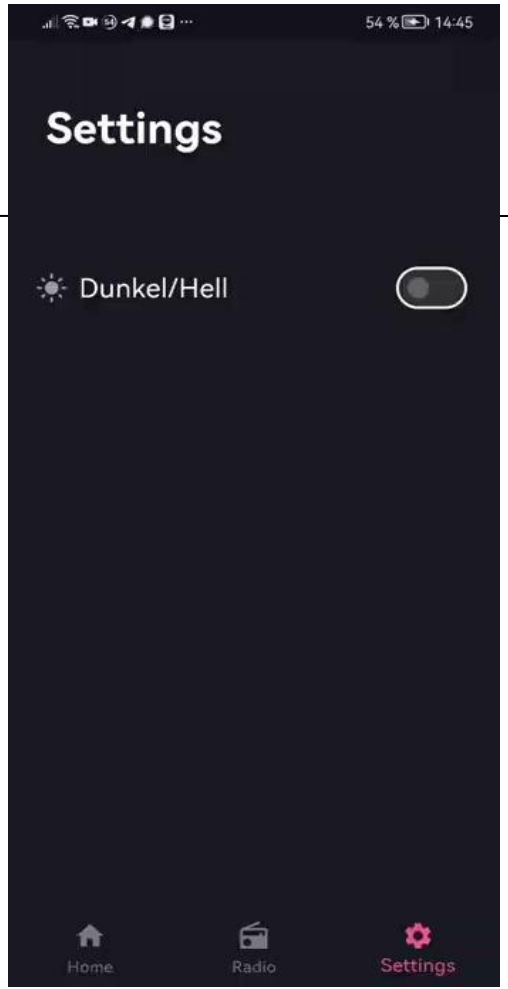


Radio-App Radioscreen

- Fokus auf aktuelles Radio
- Anzeigen von Metadaten
- Vor- und zurückschalten durch Radio-Liste

Radio-App Settings

- Auswahl von Dark- oder Lightmode

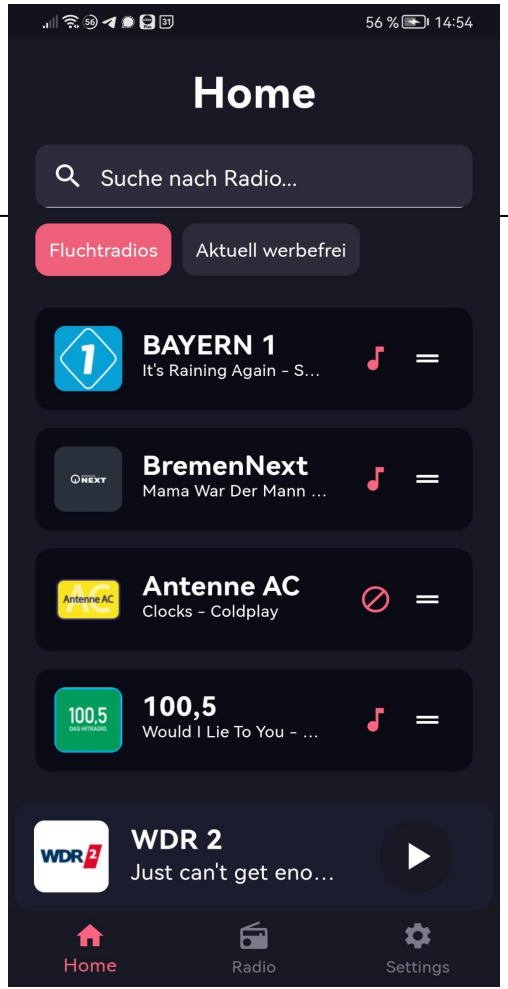


Anforderungen

- Radio-App
- **Konfiguration der Vorlieben**
- Werbung blockieren

Konfiguration der Vorlieben

- Auswahl von Fluchtradios (Favoriten)
- Ändern der Priorität der Fluchtradios



Konfiguration der Vorlieben

- Auswahl von Fluchtradios (Favoriten)
- Ändern der Priorität der Fluchtradios

Anforderungen

- Radio-App
- Konfiguration der Vorlieben
- **Werbung blockieren**

Werbung blockieren

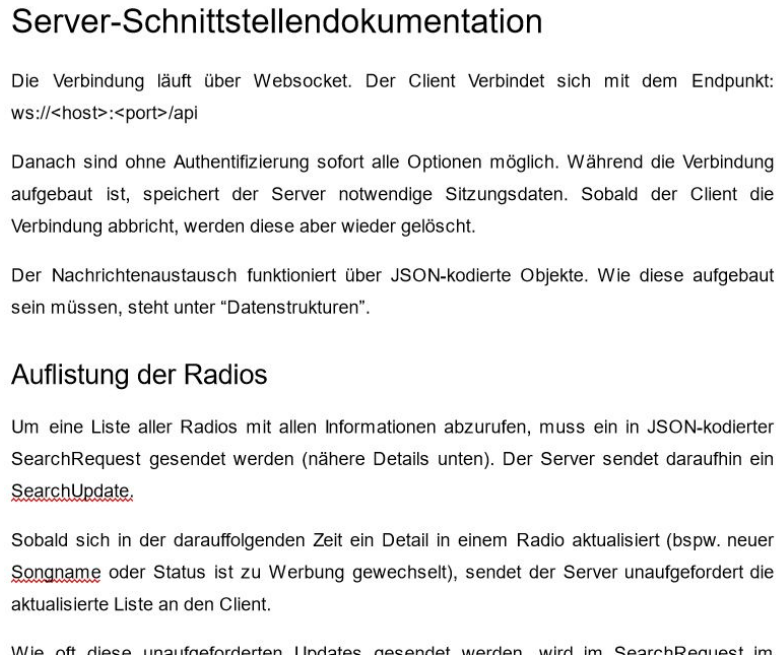
- Server API
- Zeitumschaltung
- Acoustic Fingerprinting

Werbung blockieren

- **Server API**
- Zeitumschaltung
- Acoustic Fingerprinting

Server API

- Protokoll: **WebSocket**
- Near-Real-Time Updates beim Client
- Features:
  - Auflistung der Radios
  - Guidance durch die Radios



Server-Schnittstellendokumentation

Die Verbindung läuft über Websocket. Der Client Verbindet sich mit dem Endpunkt: ws://<host>:<port>/api

Danach sind ohne Authentifizierung sofort alle Optionen möglich. Während die Verbindung aufgebaut ist, speichert der Server notwendige Sitzungsdaten. Sobald der Client die Verbindung abbricht, werden diese aber wieder gelöscht.

Der Nachrichtenaustausch funktioniert über JSON-kodierte Objekte. Wie diese aufgebaut sein müssen, steht unter “Datenstrukturen”.

Auflistung der Radios

Um eine Liste aller Radios mit allen Informationen abzurufen, muss ein in JSON-kodierter SearchRequest gesendet werden (nähere Details unten). Der Server sendet daraufhin ein SearchUpdate.

Sobald sich in der darauffolgenden Zeit ein Detail in einem Radio aktualisiert (bspw. neuer Songname oder Status ist zu Werbung gewechselt), sendet der Server unaufgefordert die aktualisierte Liste an den Client.

Wie oft diese unaufgeforderten Updates gesendet werden, wird im SearchRequest im

Mehr dazu in der Dokumentation

Werbung blockieren

- Server API
- **Zeitumschaltung**
- Acoustic Fingerprinting

Zeitumschaltung

- Analyse jedes einzelnen Radios
- Hinterlegung der Werbezeiten in Datenbank
- Separater Thread kümmert sich um Status des Radios
- Problem: Nie auf die Sekunde genau

Erkenntnisse über Werbungen

Radio	Werbezeiten	Werbelänge [Sekunden]	Stündliche Variation	Jingle vorhanden?	Weiteres
WDR2	6 - 22 Uhr oder bis 20 Uhr	10-60	nicht exakt	Ja (Anfang)	1-2 Werbeblöcke pro Stunde
1Live	6 - 22 Uhr	10-60	1-3 Minuten	Ja (Anfang)	
100,5 Hitradio	immer	10-60	unregelmäßig	Ja (Anfang)	
AntenneAC	immer	10-60	-	-	
bigFM	immer	60-120	unregelmäßig	Ja (Anfang und Ende)	Oft eigene Werbung

Zeitschaltung

Die Zeiten für Werbung der einzelnen Radios werden von uns händisch ermittelt (siehe oben "Erkenntnisse über Werbungen").

Diese werden dann in eine Tabelle in der Datenbank eingefügt.

Ein separater Thread stellt eine Anfrage an die Datenbank, wann das nächste Mal ein Wechsel von Werbung auf Musik, bzw. Musik auf Werbung ist und legt sich bis dahin schlafen. Beim Aufwachen werden dann alle Clients benachrichtigt, die benachrichtigt werden müssen.

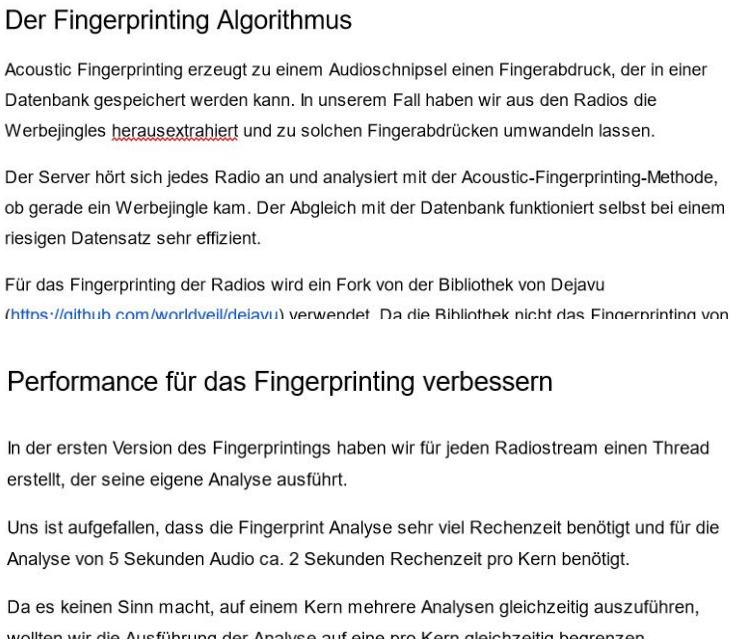
Wurde nun vom Fingerprinting vollständig abgelöst.

Werbung blockieren

- Server API
- Zeitumschaltung
- **Acoustic Fingerprinting**

Acoustic Fingerprinting

- Extraktion der einzelnen Jingles
- Jedes Radio wird von einem dediziertem Thread gehört
- Worker Threads vergleichen Audiospur mit Jingles
- Löst Zeitschaltung vollständig ab



Der Fingerprinting Algorithmus

Acoustic Fingerprinting erzeugt zu einem Audioschnipsel einen Fingerabdruck, der in einer Datenbank gespeichert werden kann. In unserem Fall haben wir aus den Radios die Werbejingles herausextrahiert und zu solchen Fingerabdrücken umwandeln lassen.

Der Server hört sich jedes Radio an und analysiert mit der Acoustic-Fingerprinting-Methode, ob gerade ein Werbejingle kam. Der Abgleich mit der Datenbank funktioniert selbst bei einem riesigen Datensatz sehr effizient.

Für das Fingerprinting der Radios wird ein Fork von der Bibliothek von Dejavu (https://github.com/worldveil/dejavu) verwendet. Da die Bibliothek nicht das Fingerprinting von

Performance für das Fingerprinting verbessern

In der ersten Version des Fingerprintings haben wir für jeden Radiostream einen Thread erstellt, der seine eigene Analyse ausführt.

Uns ist aufgefallen, dass die Fingerprint Analyse sehr viel Rechenzeit benötigt und für die Analyse von 5 Sekunden Audio ca. 2 Sekunden Rechenzeit pro Kern benötigt.

Da es keinen Sinn macht, auf einem Kern mehrere Analysen gleichzeitig auszuführen, wollten wir die Ausführung der Analyse auf eine pro Kern gleichzeitig begrenzen

Mehr dazu in der Dokumentation

Inhaltsverzeichnis


DAS IST DOCH
ALLES NEULAND FÜR MICH!

Herausforderungen: Frontend

- Verlust eines Teammitgliedes
- Anbindung des Servers
  - Exception konnte nicht abgefangen werden
  - Nutzung einer einzigen Verbindung
- Globale Datenbereitstellung
- Serverausfälle und Vermutung des Fehlers im Frontend
- Light-Mode

Herausforderungen

Frontend

Kommunikation mit dem Server

Die wahrscheinlich größte Herausforderung stellte für uns die Programmierung der Serverschnittstelle dar. Es galt zunächst große Wissenslücken zu schließen, eine geeignete Bibliothek zu finden und diese zu verstehen. Während der Implementierung traten ständig Exceptions auf, bei denen es uns zuerst nicht gelungen ist, sie abzufangen. Dies hat uns großes Kopfzerbrechen bereitet.

Des Weiteren ist uns erst relativ spät klar geworden, dass für die Bereitstellung der Liste und des spielbaren Radios jeweils eine Verbindung zum Server aufgebaut werden muss. Lange Zeit erfolgte beides über die gleiche Verbindung, was ständig zu Fehlern führte. Die Ursache des Problems zu finden, hat lange Zeit in Anspruch genommen.

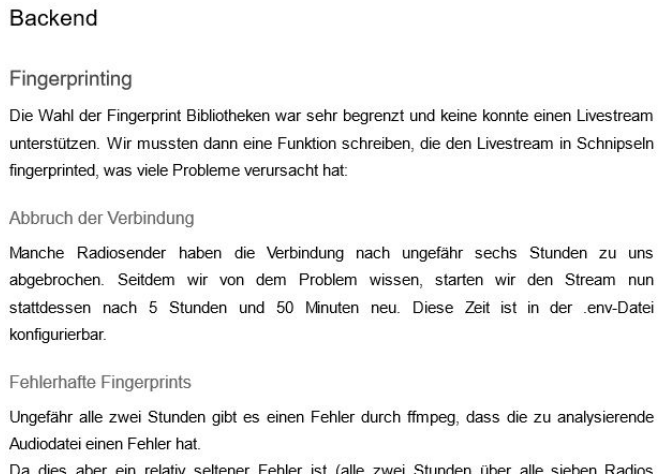
Datenbereitstellung

Damit die vom Server empfangenen Daten global in der App zur Verfügung gestellt werden konnten, mussten sogenannte Provider verwendet werden. Anfangs entschieden wir uns für

Mehr dazu in der Dokumentation

Herausforderungen: Backend

- Abbruch der Verbindung durch Radioanbieter
- Doppelte Fingerprints
- Performance Probleme
- Zwei verschiedene Datenbanken
- Mehrere APIs für Metadaten



Backend

Fingerprinting

Die Wahl der Fingerprint Bibliotheken war sehr begrenzt und keine konnte einen Livestream unterstützen. Wir mussten dann eine Funktion schreiben, die den Livestream in Schnipseln fingerprinted, was viele Probleme verursacht hat:

Abbruch der Verbindung

Manche Radiosender haben die Verbindung nach ungefähr sechs Stunden zu uns abgebrochen. Seitdem wir von dem Problem wissen, starten wir den Stream nun stattdessen nach 5 Stunden und 50 Minuten neu. Diese Zeit ist in der .env-Datei konfigurierbar.

Fehlerhafte Fingerprints

Ungefähr alle zwei Stunden gibt es einen Fehler durch ffmpeg, dass die zu analysierende Audiodatei einen Fehler hat.

Mehr dazu in der Dokumentation

Inhaltsverzeichnis

Erreichte Ziele

- Software
- Adblock
- Auswahlmöglichkeiten
- Methodik

Erreichte Ziele

- **Software**
- Adblock
- Auswahlmöglichkeiten
- Methodik

Erreichte Ziele

- **Software**
  - MVP: Android-App
  - Und: Windows-App

- Adblock
- Auswahlmöglichkeiten
- Methodik

Erreichte Ziele

- Software
- **Adblock**
- Auswahlmöglichkeiten
- Methodik

Erreichte Ziele

- Software
- **Adblock**
  - MVP: Zeitgesteuerte Umschaltung
  - Und: Erkennung der Werbung durch Acoustic Fingerprinting
- Auswahlmöglichkeiten
- Methodik

Erreichte Ziele

- Software
- Adblock
- **Auswahlmöglichkeiten**
- Methodik

Erreichte Ziele

- Software
- Adblock
- **Auswahlmöglichkeiten**
  - MVP: Vorliebe „Musik“, „Nachrichten“, „Lokales“
- Methodik

Erreichte Ziele

- Software
- Adblock
- Auswahlmöglichkeiten
- **Methodik**

Erreichte Ziele

- Software
- Adblock
- Auswahlmöglichkeiten
- **Methodik**
  - MVP: direktes Umschalten zwischen den Stationen

Selbstgesteckte Ziele, die wir erreicht haben

- Ausführliche Dokumentation
- Docker
- Gutes Design
- Logs für Analyse
- Debug-Oberfläche

Inhaltsverzeichnis

Ausblick

- Businessmodell
- Skalierbarkeit
- Testing
- Abhängigkeiten

Ausblick

- **Businessmodell**
- Skalierbarkeit
- Testing
- Abhängigkeiten

Businessmodell

Da der Server nicht kostenlos ist, muss man sich langfristig über eine Finanzierung Gedanken machen. Hier sind mehrere Ansätze, die wir uns überlegt haben:

Analyse der Konkurrenz

Der Server kann die Abläufe der Radios loggen und die klassifizierte Werbung einer Firma zuordnen. Die Konkurrenz der Firmen, die Werbung ausstrahlen lässt, ist möglicherweise daran interessiert, wann und wie oft von welcher Firma Werbung ausgestrahlt wird.

Kostenpflichtige App

Die App könnte man im Google Play Store und App Store für einen angemessenen Preis veröffentlichen. Will man werbefreies Radio hören, muss man also einmalig diese App kaufen.

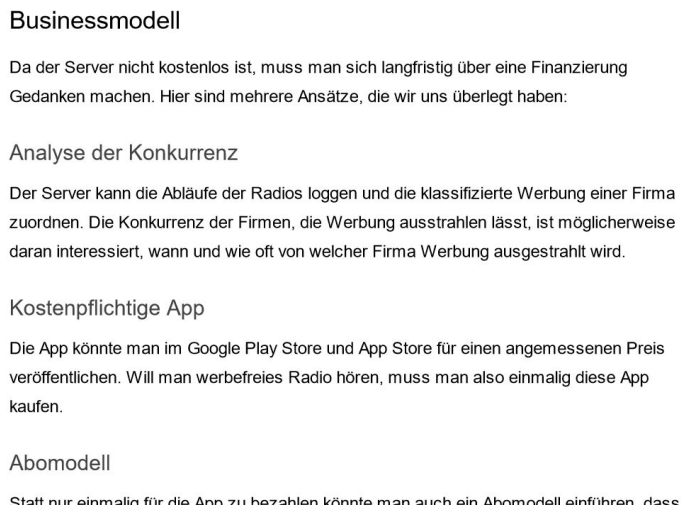
Abomodell

Statt nur einmalig für die App zu bezahlen könnte man auch ein Abomodell einführen, dass

Mehr dazu in der Dokumentation

Businessmodell

- Analyse der Konkurrenz
- Kostenpflichtige App
- Abomodell
- Daten der Benutzer sammeln und verkaufen
- Werbebanner in der App anzeigen



Businessmodell

Da der Server nicht kostenlos ist, muss man sich langfristig über eine Finanzierung Gedanken machen. Hier sind mehrere Ansätze, die wir uns überlegt haben:

Analyse der Konkurrenz

Der Server kann die Abläufe der Radios loggen und die klassifizierte Werbung einer Firma zuordnen. Die Konkurrenz der Firmen, die Werbung ausstrahlen lässt, ist möglicherweise daran interessiert, wann und wie oft von welcher Firma Werbung ausgestrahlt wird.

Kostenpflichtige App

Die App könnte man im Google Play Store und App Store für einen angemessenen Preis veröffentlichen. Will man werbefreies Radio hören, muss man also einmalig diese App kaufen.

Abomodell

Statt nur einmalig für die App zu bezahlen könnte man auch ein Abomodell einführen, dass

Mehr dazu in der Dokumentation

Ausblick

- Businessmodell
- **Skalierbarkeit**
- Testing
- Abhängigkeiten

Skalierbarkeit: Frontend

- Liste für Homescreen sollte in Blöcken angefragt werden
- Filterung aktuell beim Client
  - Zukünftig serverseitig
- Nur eine Verbindung zum Server pro Client

Frontend

Aktuell wird die gesamte Liste der Radios an den Client übertragen. Dies mag bei einer Liste mit Anzahl < 100 noch zuverlässig funktionieren, jedoch ist dieser Ansatz nicht skalierbar. Besser wäre es, eine Pagination hinzuzufügen, sodass die Einträge in kleinen Blöcken angefragt werden können. Im Sinne von: “Gib mir Einträge 0 bis 50”, man scrollt weiter: “Gib mir Einträge 50 bis 100”.

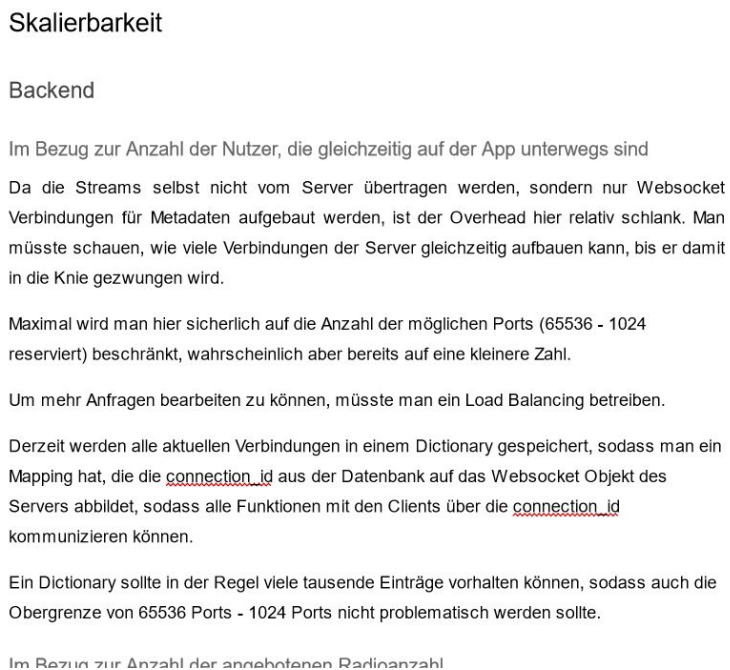
Augenblicklich wird die Filterung vom Client übernommen. Beim Steigern der Radioanzahl könnte dies zu Performance-Problemen führen. Der Server hat intern bereits eine Struktur zur Filterung, die erweitert und als API zur Verfügung gestellt werden müsste. Dies müsste das Frontend sowie das Backend noch nachträglich hinzufügen.

Die App baut derzeit zwei Verbindungen zum Server auf. Dies beansprucht unnötig viele Ressourcen vom Server. Besser wäre es, sämtliche Kommunikation über eine Verbindung laufen zu lassen.

Mehr dazu in der Dokumentation

Skalierbarkeit: Backend

- Ein Load-Testing wäre interessant
- API ist momentan single threaded
- Verbindungs-IDs werden vollständig im RAM gehalten
- Fingerprinting sehr rechenintensiv



Skalierbarkeit

Backend

Im Bezug zur Anzahl der Nutzer, die gleichzeitig auf der App unterwegs sind

Da die Streams selbst nicht vom Server übertragen werden, sondern nur Websocket Verbindungen für Metadaten aufgebaut werden, ist der Overhead hier relativ schlank. Man müsste schauen, wie viele Verbindungen der Server gleichzeitig aufbauen kann, bis er damit in die Knie gezwungen wird.

Maximal wird man hier sicherlich auf die Anzahl der möglichen Ports (65536 - 1024 reserviert) beschränkt, wahrscheinlich aber bereits auf eine kleinere Zahl.

Um mehr Anfragen bearbeiten zu können, müsste man ein Load Balancing betreiben.

Derzeit werden alle aktuellen Verbindungen in einem Dictionary gespeichert, sodass man ein Mapping hat, die die connection_id aus der Datenbank auf das Websocket Objekt des Servers abbildet, sodass alle Funktionen mit den Clients über die connection_id kommunizieren können.

Ein Dictionary sollte in der Regel viele tausende Einträge vorhalten können, sodass auch die Obergrenze von 65536 Ports - 1024 Ports nicht problematisch werden sollte.

Im Bezug zur Anzahl der angebotenen Radioanzahl

Mehr dazu in der Dokumentation

Ausblick

- Businessmodell
- Skalierbarkeit
- **Testing**
- Abhängigkeiten

Testing

Die Korrektheit der aktuellen Version für das Fingerprinting, sowie das Erhalten der Metadaten wurde durch Real Time Testing verifiziert. Um eine bessere Robustheit des Programms sicherzustellen, wäre es sinnvoll, ein Testing Framework zu verwenden, bei dem man mit Unit-Tests arbeitet, um bspw. die Funktionen der API im Backend gründlich zu testen, ohne dass das Frontend Fehler bekommt, für die sie gar nicht verantwortlich sind.

Tests, die ständig überprüfen, ob die Metadaten und Werbestatus aktualisiert werden. Zuletzt gab es immer wieder das Problem, dass die Metadaten irgendwann aufgehört haben, sich zu aktualisieren.

Ausblick

- Businessmodell
- Skalierbarkeit
- Testing
- **Abhängigkeiten**

Abhängigkeiten

API

Wir nutzen für unsere Metadaten die folgenden APIs:

https://prod.radio-api.net/stations/now-playing?stationIds=<station_id1>,<station_id2>,...

https://api-prod.nrwlokalradios.com/playlist/current?station=<station_id>

Diese nutzen wir ungefragt und diese wurden außerdem nicht nach außen hin für die Öffentlichkeit dokumentiert. Es kann also sein, dass sich die APIs jederzeit einfach ändern können.

URLs

Wir haben einmalig die Stream URLs der von uns angebotenen Radios herausgesucht. Ändern diese sich bei dem Anbieter, funktioniert das Radio bei uns nicht mehr. Dadurch müssen bei der Weiterverwendung des Projektes die Radio URLs je nachdem gepflegt werden. Bei den URLs ist zu beachten, dass der Content-Type durchgängig "audio/mpeg" ist, da unsere Fingerprint-Bibliothek nur mpeg unterstützt.

Mehr dazu in der Dokumentation

Inhaltsverzeichnis

Einzelaufwände: Ahmad Fadi Aljabri (Frontend)

- Settingsscreen
- Persistierung
- Light/Dark Mode
- Fehlermeldungen

- Erstellung des Mockups für den Setting Screen
- Persistierung
  - Favoriten
  - zuletzt gehörtes Radio
  - Settings
  - Dark/Light mode
- Benachrichtigung wenn ein Radio angeklickt wird, wo Werbung läuft
  - Programmierung eines Dialogs
- Programmierung des Setting Screens

Mehr dazu in der Dokumentation

Einzelaufwände: Nicolas Harrje (Frontend)

- Radioscreen
- Navigationsleiste
- Anbindung des Servers
- Priorisierung

Nicolas Harrje

- Erstellung des Mockups für den Radioscreen
- Programmierung der Navigationsleiste
- Programmierung des Radioscreens
  - Funktionalität der Steuerungsbuttons
  - Darstellung der Metadaten
- Anbindung des Servers
  - Verbindungsaufbau zum Server
  - Daten für Listendarstellung im Homescreen abfragen
  - Implementierung des Providers für die Listendarstellung
- Priorisierung
  - Initialisierung der Prioritäten

Mehr dazu in der Dokumentation

Einzelaufwände: Nils Stegemann (Frontend)

- Homescreen
- Anbindung des Servers
- Audiowiedergabe
- Auto-Scrolling Animation
- Persistierung

Nils Stegemann

- Erstellung des Mockups für den Homescreen
- Programmierung des Homescreens
  - Filteroptionen
    - Textuelle Suche
    - Fluchtradios
    - Aktuell Werbefrei
  - Liste aller verfügbaren Radios
    - Indikator Werbung / keine Werbung
    - Vergabe der Favoriten
    - Darstellung der Metadaten
  - Darstellung des gespielten Radios mit Play Button zum starten und stoppen des Audiostreams.
- Anbindung des Servers
  - Verbindungsaufbau zum Server

Mehr dazu in der Dokumentation

Einzelaufwände: Chris Henkes (Backend)

- Zeitbasierte Umschaltung zwischen Radios
- Client benachrichtigen
- Radioumschaltung durch Werbejingle
- Metadaten der Radios

Chris Henkes

- Zeitbasierte Umschaltung der Radios zwischen Werbung und Musik
- Verbesserung der Performance der zeitbasierten Umschaltung
- Thread basierte Notifications an Clients senden
  - für das aktuelle aktive Radio
  - für alle in der App vorhandenen Radios
- Aktuelle Metadaten(Song und Interpret) der Radios mit Hilfe der Radio.de API erhalten und an Client weiterleiten
- Umschaltung zu Werbung der Radios anhand der jeweiligen Werbejingles
  - Bei Radios mit Werbe end Jingles, mit diesem zurück zum Radio schalten
  - Bei Radios ohne Werbe end Jingles, mit einer festgelegten 'Werbezeit' zurück schalten

Mehr dazu in der Dokumentation

Einzelaufwände: Kaan Yazici (Backend)

- Prototyp Client (Python)
- Fingerprinting + Serveraufnahme
- Jingle Extraktion
- Statistiken

Kaan Yazici

- Für Tests: Erstellung eines Konsolenprogramms, das über ähnliche, aber vereinfachte Funktionen wie die vom Frontend entwickelte App verfügt
- Korrekte Fehlerbehandlung in der Server Schnittstelle (Version 1)
- Analyse der Radios
  - Antenne AC, Radio RST, BAYERN 1, 100.5 Das Hitradio, 1Live, WDR2, BigFM
  - Aufnahme der Radios
  - Extraktion der Zeiten, in denen Werbung abgespielt wird
  - Extraktion der Zeiten, in denen Nachrichten abgespielt werden
  - Siehe Anlage “Recherche”
- Recherche zu einer geeigneten Python Bibliothek, mit der wir Acoustic Fingerprintings von Jingles erstellen und diese Jingles möglichst präzise aus dem Livestream erkennen können
- Extraktion und Nachbearbeitung der einzelnen Werbejingles, damit diese mit der Fingerprinting Technologie erkannt werden können

Mehr dazu in der Dokumentation

Einzelaufwände: Max Breuer (Backend)

- Datenbank Design
- ORM Implementierung
- Logging der Metadaten + Adtimes
- Funktionsdokumentation

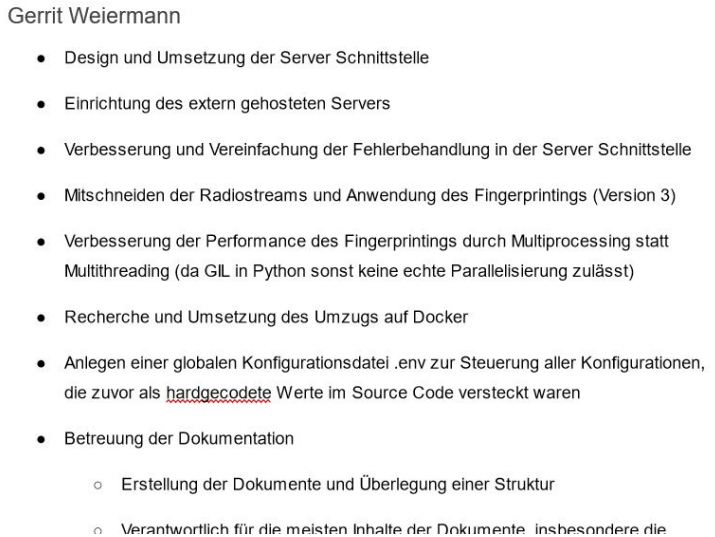
Maximilian Breuer

- Recherche zur Datenbank
- Recherche zu einem geeigneten Object Relational Mapping Framework (ORM)
- Design der Datenbank
- Umsetzung der Datenbank im ausgewählten ORM in Python
- Um die zwei Datenbanken MySQL und PostgreSQL auf eine Datenbank zu verschmelzen
    - Recherche zur Migration der PostgreSQL Datenbank auf MySQL
    - Umsetzung der Migration von PostgreSQL auf MySQL (leider gescheitert)
    - Recherche den MySQL Treiber der Fingerprinting Bibliothek auf einen PostgreSQL Treiber umschreiben
    - Umsetzung der Migration von MySQL auf PostgreSQL (leider gescheitert)
- Erstellung von vielen Hilfsfunktionen, die Lese-, Schreib- und Löschaktionen auf der PostgreSQL Datenbank vornehmen, damit Statements wiederverwendbar werden

Mehr dazu in der Dokumentation

Einzelaufwände: Gerrit Weiermann (Backend)

- Umsetzung der ersten Version der API
- Migration zu Docker
- Anlegen einer zentralen Konfigurationsdatei
- Betreuung der Dokumentation im Backend



Gerrit Weiermann

- Design und Umsetzung der Server Schnittstelle
- Einrichtung des extern gehosteten Servers
- Verbesserung und Vereinfachung der Fehlerbehandlung in der Server Schnittstelle
- Mitschneiden der Radiostreams und Anwendung des Fingerprintings (Version 3)
- Verbesserung der Performance des Fingerprintings durch Multiprocessing statt Multithreading (da GIL in Python sonst keine echte Parallelisierung zulässt)
- Recherche und Umsetzung des Umzugs auf Docker
- Anlegen einer globalen Konfigurationsdatei .env zur Steuerung aller Konfigurationen, die zuvor als hardgecodete Werte im Source Code versteckt waren
- Betreuung der Dokumentation
  - Erstellung der Dokumente und Überlegung einer Struktur
  - Verantwortlich für die meisten Inhalte der Dokumente, insbesondere die

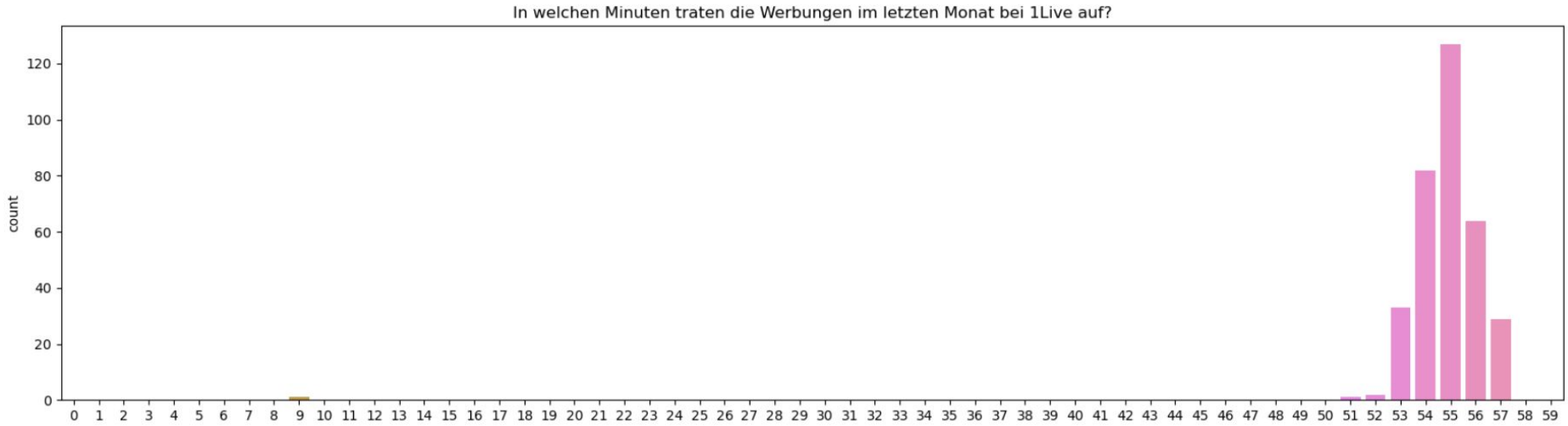
Mehr dazu in der Dokumentation

Inhaltsverzeichnis

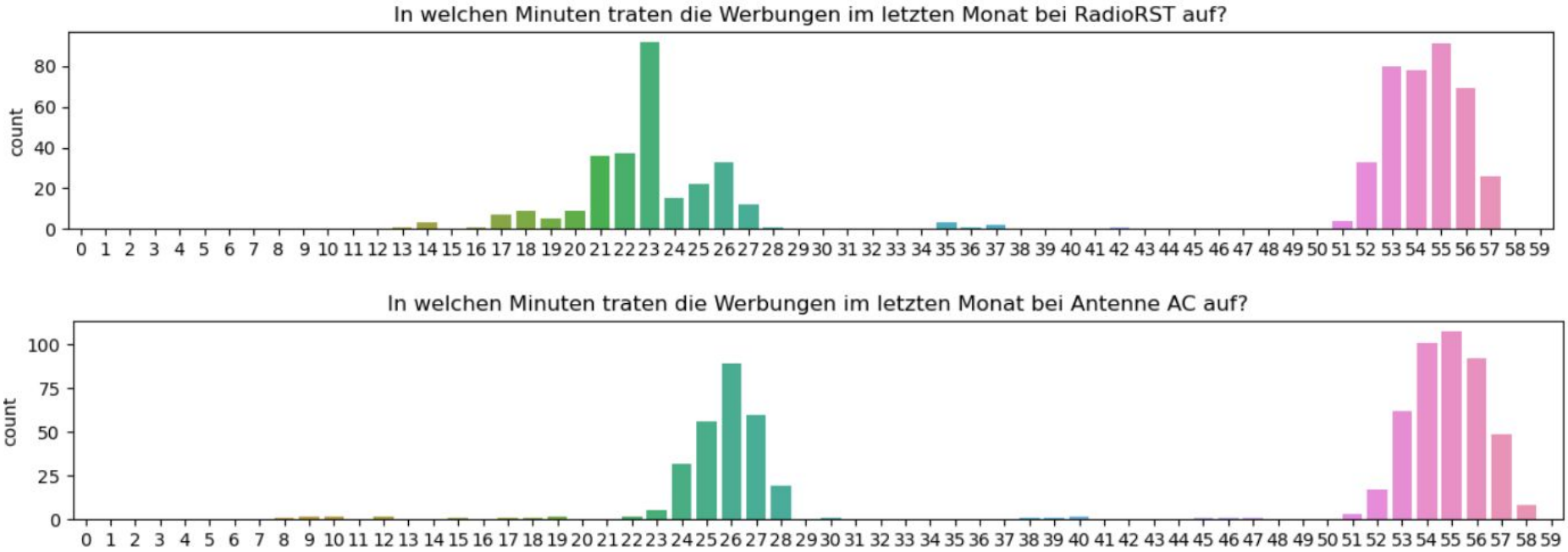
Statistiken

Wir haben Daten gesammelt.

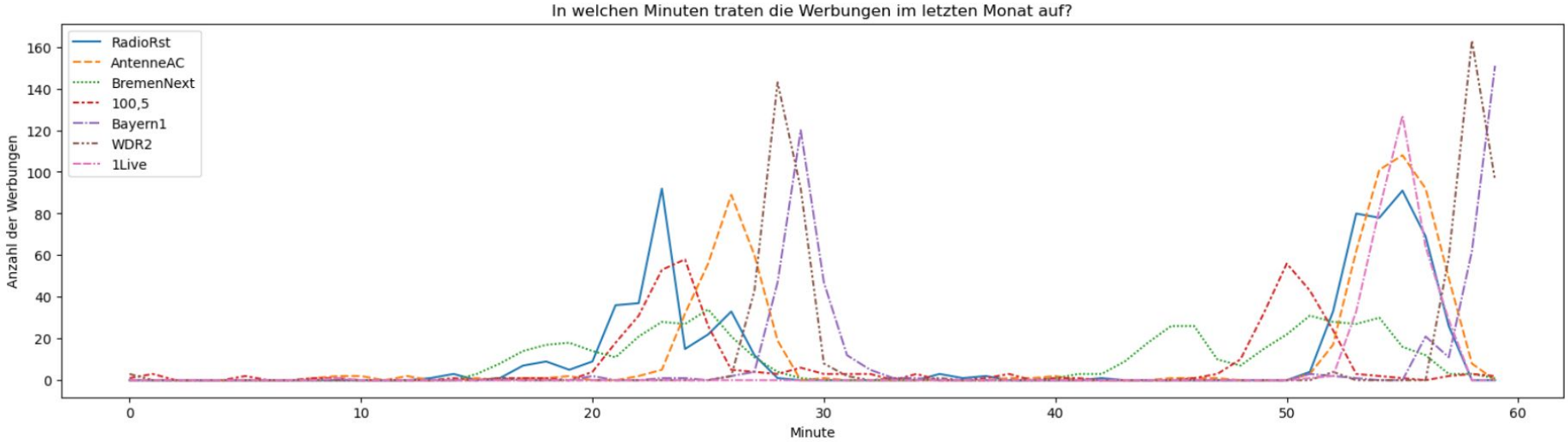
Wann startet bei 1Live die Werbung?



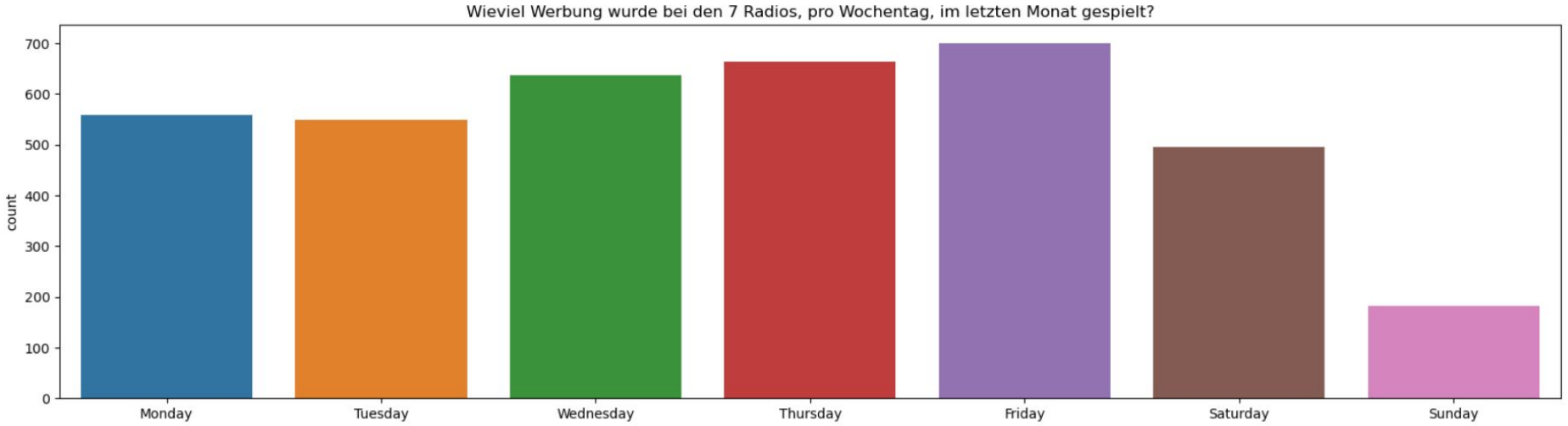
Wann startet bei den RadioRST und Antenne AC die Werbung?



Wann startet generell die Werbung?



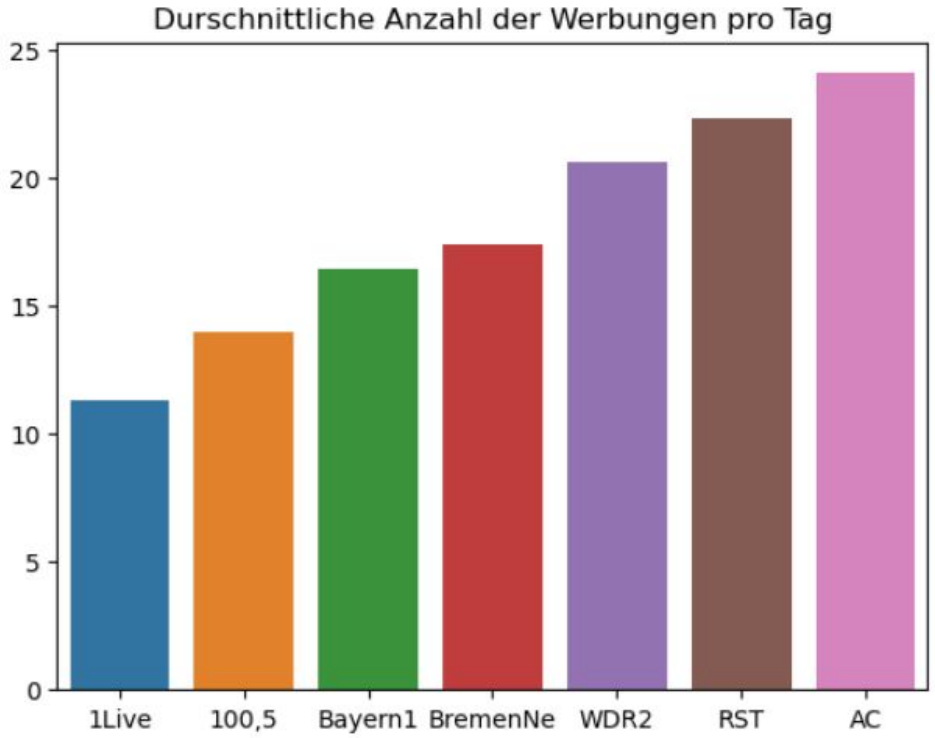
An welchen Tagen wird die meiste Werbung geschaltet?



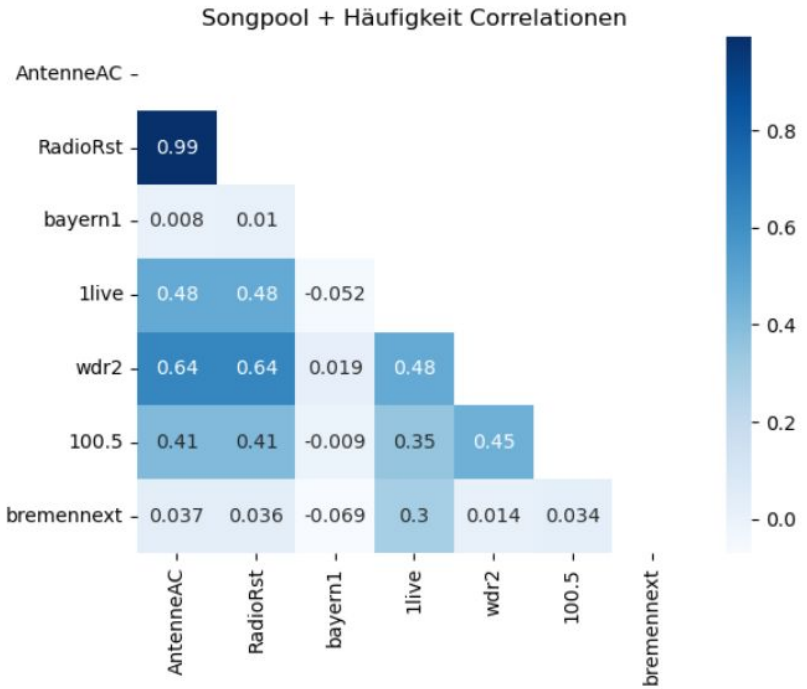
Welches Radio spielt am meisten Werbung?

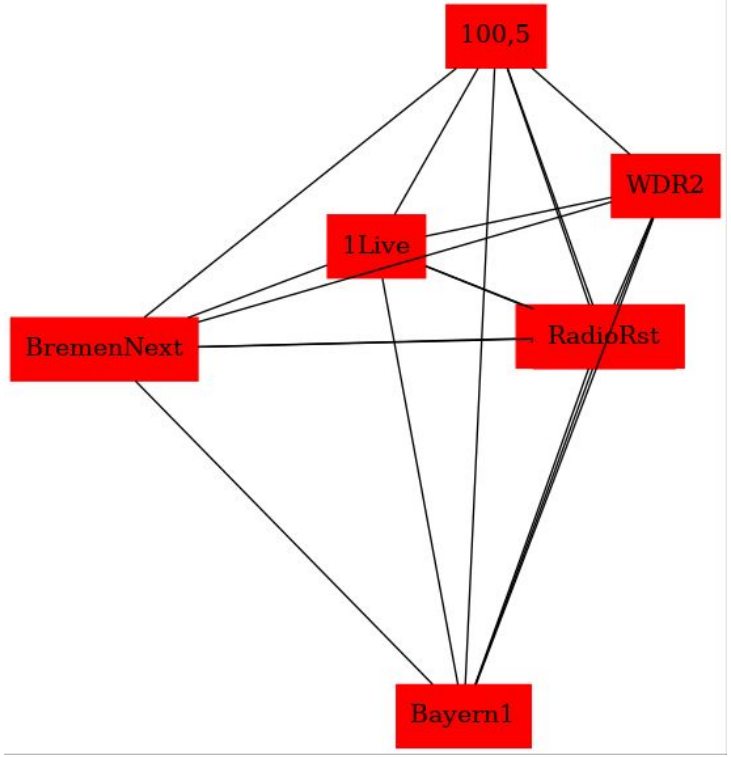
1. 1Live
2. WDR 2
3. Bayern1
4. 100,5
5. RadioRST
6. AntenneAC
7. BremenNext

Welches Radio spielt am meisten Werbung?



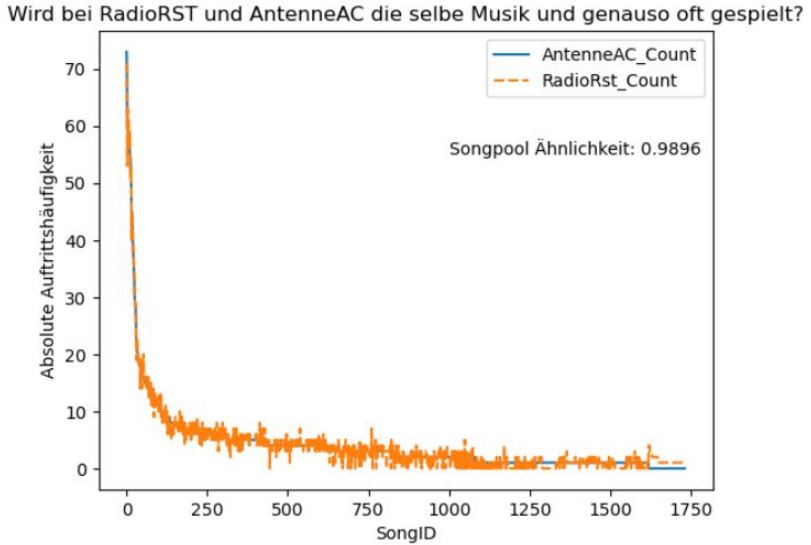
Wie ähnlich sind sich die Radios?



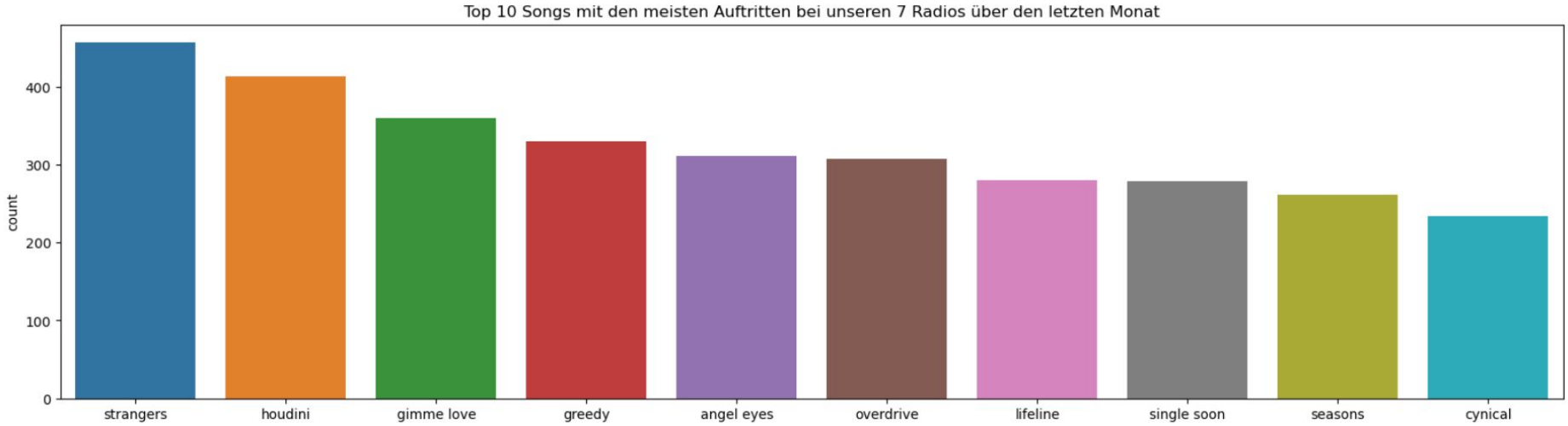


Charts der NRW Lokalradios

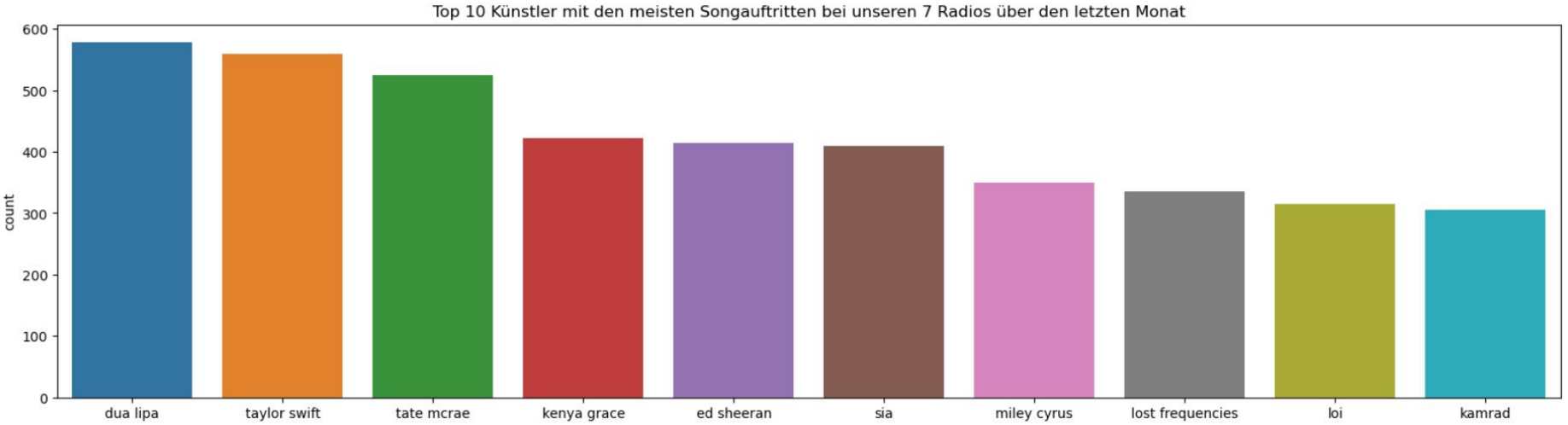
1	Lifeline - Glockenbach	73 Mal
2	Strangers - Kenya Grace	70 Mal
3	Houdini - Dua Lipa	61 Mal
4	Dreaming - Marshmello	59 Mal
5	Where you are - Lost Frequencies	59 Mal



Song Charts der sieben Radios



Künstler Charts der sieben Radios



Inhaltsverzeichnis

Livedemo

FH Aachen
Fachbereich 5
Team IP Radio Adblocker