Projektbreicht Smart Music Player

Anton Bracke
Jan Eberlein
Tom Calvin Haak
Julian Hahn
Nick Loewecke

26. Januar 2021

Inhaltsverzeichnis

1 Einleitung

1.1 Unternehmen

TODO: was macht macio aus

1.2 Projektidee

Im Rahmen des Projekt Informatik möchte Macio ihr Portfolio im IoT-Bereich erweitern, sowie ihren Empfangsraum im Standort Kiel verschönern. Hierfür soll eine smarte Spielzeug-Box gebaut werden. Smarte Spielzeuge gibt es im kommerziellen Bereich viele, daher soll dieses Projekt eine Open-Source-Alternative schaffen.
Genauer handelt es sich um eine Musik-Box, die NFC-Chips lesen und Spotify Connect unterstützen soll. Auf die Box können dann Spielzeuge (z.B. in Form von kleinen Figuren) mit integrierten NFC-Chips gestellt werden, um spezifische Musik abspielen zu lassen. Die Musik wird von Spotify-Connect über eine bereits bestehende Musik-Anlage abgespielt. Falls es im Rahmen des Projektes möglich ist, sollen die Nutzer in der Lage sein, zwischen verschiedenen Musikanbietern zu wechseln. NFC-Chips und die zugehörige Musik sollen über ein Web-basierte Benutzeroberfläche konfiguriert werden können. Diese Benutzeroberfläche soll von der Box ausgeliefert und primär für Smartphone-Bedienung gestaltet werden. Da es sich um ein Open Source Projekt mit entsprechender Lizenz handelt, muss auch eine aussagekräftige, öffentliche Dokumentation verfasst werden. Macio stellt die benötigte Hardware zur Verfügung und unterstützt bei technischen Fragen.

1.2.1 Minimal Requirements

1. NFC-Tags lesen, schreiben und entschlüsseln
2. Mit Spotify Connect verbinden und arbeiten
3. Responsive UI konzeptionieren und umsetzen
4. Aussagekräftige Dokumentation mit Benutzerhandbuch

1.2.2 Stretch Goals

1. Sound Wiedergabe auf der Box selbst
2. Unterstützung anderer Musikdienste / Plugin-Subsystem
3. 3D-Modellierung und Print einer passenden Box
4. Cloud-Anbindung der Box, Auslieferung des UI aus der Cloud

2 Projektplanung

2.1 Projekt Management

TODO: ticket pool in github, gemeinsame Absprache (im team und mit Kunden) was in den aktuellen Milestone kommen soll, alles zentral, 3x daily standup,

gemeinsamer google calendar, gemeinsamer Discord Server (für spontaneres und informelleres gemeinsames Arbeiten), interne team evaluation und regelmäßiges feedback

2.2 Projektrisiken

TODO: Welche Riskiken hatten wir + Risikomatrix? lohnt sich dieser Abschnitt?

2.3 Phasenplanung

2.3.1 Projektstartphase

TODO: Github aufsetzen, Tickets befüllen, Anforderungen verstehen, Team einarbeiten in Technologien

2.3.2 Realisierungsphase

TODO: Iteratives Arbeiten in Meilensteinen/Sprint mit Länge ca. 2 Wochen. Coden etc. Jedes Arbeitspaket wird einzeln getestet und von mindestens zwei anderen Personen reviewt (Technische Funktionalität wird automatisch getestet)

2.3.3 Vermarktungsphase

TODO: Testsachen aus Code nehmen, Anleitung schreiben, Informationen veröffentlichen

2.4 Projektstrukturplan

TODO: einzelne subsubsections über frontend, backend, hardware, devops, und dann erklären was das beinhaltet?

2.5 Beispielhafte Arbeitspakete

TODO: 1-2-3 Beispielhafte Arbeitspakete ausarbeiten? Am besten auch welche, die von anderen abhängig

2.6 Netzplan

TODO: wie hängt alles zusammen mit approx Zeit, wobei wir die Zeit vielleicht weglassen?

3 Machbarkeitsstudie

3.1 NFC Tag

3.1.1 lesen

Um mit NFC Tags arbeiten zu können, müssen diese auch entschlüsselt bzw. gelesen werden können. Hierfür ist ein Hardware NFC-Reader notwendig, der

die Daten ausliest und an die Box kommuniziert. Dieser emuliert dafür Keyboard Eingaben, die die Tag-IDs darstellen. *Evdev*, ein Kernel Modul von Linux, könnte dann zum Abgreifen dieser Keyboard-Eingaben genutzt werden. Mit *node-evdev*[1] kann *evdev* auch mit Node genutzt werden. Mit dem Fork[2] von Anbraten wird zusätzlich der Raspberry Pi und die Typescript Unterstützung zur Verfügung gestellt.

3.1.2 schreiben

Ein NFC-Tag hat generell eine feste ID. Um weitere Daten auf einen NFC-Tag schreiben zu können, benötigt der NFC-Tag also einen eigenen Speicher. Ist dieser vorhanden, können dort z.b. Kontaktdaten hinterlegt werden. Werden diese dann von einem Smartphone gelesen, öffnet sich die Kontakte-App und der auf dem NFC-Tag gespeicherte Kontakt kann abgespeichert werden. Dafür wäre einerseits ein spezieller NFC-Reader, der auch schreiben kann, sowie eine spezielle Library notwendig.

3.2 Raspberri Pi

3.2.1 Docker Integration

Auf einem Raspberri Pi Docker zu installieren und zum Laufen zu bringen wird in vielen Anleitungen online beschrieben[3].

3.2.2 Öffentlich zugängliches Web Interface

Auf einem Raspberri Pi könnte eine Web Anwendung gehosted werden, welche die allgemeine Web Anwendung für alle Boxen darstellen soll. Um diese Web Anwendung von außerhalb des eigenen Netzwerkes erreichen zu können, muss innerhalb des Routers ein Port ge-forwarded werden. Dann kann der Pi und dessen Webinterface unter der öffentlichen IP xxx.xxx.xxx.xxx des Routers erreicht werden. Da sich die öffentliche IP-Adresse eines privaten Internet-Anschlusses in der Regel täglich ändert, wird zum einfachen finden der IP ein DynDns Service benötig, welcher eine feste Domain in die wechselnde IP Adresse des Routers übersetzt. Alternativ ginge es auch ohne Port-Forwarding mit nginx und ngrok [4]. Für Unerfahrene wären diese notwendingen Schritte zu Beginn etwas komplexere Thematiken. Der effektive Arbeitsaufwand hängt daher auch sehr stark von der Erfahrung der einzelnen Teammitglieder ab.

3.2.3 URL für UI festlegen

Über ein mDNS Service, der auf dem Raspberri Pi läuft, wäre es möglich für die statische Public-IP eine eigene URL anzulegen. Dafür sind verschiedene mDNS Services möglich, potentiell ist auch eine Domain notwendig.

[1] https://github.com/sdumetz/node-evdev
[2] https://github.com/anbraten/node-evdev
[3] https://phoenixnap.com/kb/docker-on-raspberry-pi
[4] https://vatsalyagoel.com/setting-up-a-public-web-server-using-a-raspberry-pi-3/

3.3 User Interface

3.3.1 Zugriff auf NFC Reader von Cloud Anwendung

Um von der App auf die Daten vom NFC-Reader der verschiedenen Boxen zuzugreifen, wäre ein zentrales Backend mit einer API sinnvoll. Der Computer der Box könnte beim Lesen eines NFC-Tag Daten über einen API Call an die Cloud Anwendung schicken, sodass die zusätzlich zu dem Backend verbundene App das gewünschte Event triggern kann.

3.3.2 Login via Spotify, Youtube, etc.

Es gibt ein Feathers Plugin, welches die Möglichkeit bietet OAuth Provider zu nutzen, um sich über andere Services wie Spotify anzumelden.

3.3.3 Gleichen Nutzer bei verschiedenen Loginvarianten wiedererkennen

Um gleiche Nutzer zu erkennen, müssten Merkmale angelegt werden, über die diese Nutzer wiedererkennbar wären. Die E-Mail wäre hierbei ein geeignetes Merkmal, da dies einzigartig ist. Über gesetzte Scopes in der OAuth Anfrage kann diese vom jeweiligen Provider mitgeliefert werden. Um die E-Mail als Wiedererkennungsmerkmal zu verwenden, muss vorausgesetzt sein, dass Nutzer immer die gleiche E-Mail bei den unterschiedlichen Providern nutzen. Dies ist aber nicht immer der Fall. Daher könnte dem (bereits eingeloggten) Nutzer die Möglichkeit gegeben werden, weitere Accounts zu dem bestehenden hinzuzufügen und entsprechend in der Datenbank zu hinterlegen.

3.3.4 Musik Artwork laden

Sollte bei der Verwendung von Spotify kein Problem sein, da zu jeder Anfrage von Titeln oder Liedern auch eine Liste von Bildern enthalten ist.[5]

3.3.5 Eigene Bilder hochladen

Eigene Bilder hochzuladen sollte möglich sein. In unserem Kontext mit Vue.js und Node.js würde das Plugin *vue-picture-input* helfen. Mit einem Axios Post könnte das Bild an das Backend gesendet werden. [6]

3.3.6 Spotify Connect Lautsprecher auswählen

Das Auswählen von einem spezifischen Spotify Connect Lautsprecher ist möglich. Über einen API Call an die Spotify API mit dem Endpunkt */v1/me/player /device* wird eine Liste von allen verbundenen Geräten geliefert. Über den Endpunkt kann ein entsprechendes Lied zum Abspielen über den jeweiligen *Spotify Connected Speaker* übergeben werden. Sollte nicht explizit ein Lautsprecher angegeben werden, so wird der zuletzt aktive genutzt. Dieser hat bei *is_ active* den Wert *true*. [7]

[5] https://stackoverflow.com/questions/10123804/retrieve-cover-artwork-using-spotify-api
[6] https://www.digitalocean.com/community/tutorials/vuejs-uploading-vue-picture-input
[7] https://developer.spotify.com/documentation/web-api/guides/using-connect-web-api/

3.3.7 Spotify Connect Lautsprecher speichern

Die Liste von verbundenen Geräten, die über einen Call an die Spotify API erhalten wird, enthält auch ein eindeutiges Feld *id*, welches sich zusammen mit einem Namen speichern lässt.

3.3.8 In der Cloud Anwendung die eigene Box auswählen / verbinden

Bei der Ersteinrichtung könnte der Nutzer über die Eingabe der MAC Adresse oder über eine andere festgelegte ID die eigene Box finden und zu seinem Account hinzufügen. Die jeweilige Box wäre dem System anschließend bekannt und könnte zum Beispiel über den vom Nutzer gewählten Namen wiedergefunden und ausgewählt werden.

3.3.9 Boxdaten über Cloud Anwendung ändern

Um die auf der Box gespeicherten Daten aus der Cloud Anwendung heraus zu ändern, könnte ein direkter Aufruf einer API, welche auf der eigenen Box läuft, genutzt werden. Um die Verbindung zu der Box aufbauen zu können, könnte diese auf dem Cloud Backend die entsprechenden Verbindungsdaten hinterlegen.

3.3.10 Unterstützung von Youtube Music

Eine Umsetzung könnte sich als umständlich erweisen, da es bisher noch keine dedizierte Youtube Music API gibt.

3.3.11 Unterstützung von Youtube

Youtube bietet die Möglichkeit, nach Videos zu suchen [8]. Um diese Videos auf dem Raspberry als Musik abzuspielen würde sich *youtube-dl* zum downloaden der Videos als *.mp3* Dateien und *omxplayer* zum Abspielen anbieten. Hierfür wäre allerdings ein entsprechender Lautsprecher am Raspberry erforderlich. Ein vergleichbares System zu Spotify Connect existiert derzeit noch nicht.

3.3.12 Unterstützung von Apple Music

Apple Music bietet hier mit deren MusicKit JS[9] eine Möglichkeit, um Musik abzuspielen.

3.3.13 Unterstützung von Deezer

Deezer lässt sich vergleichbar zu Spotify über eine API steuern.[10]

3.3.14 Unterstützung von eigener Musik (USB Stick, MicroSD Karte, Cloud)

Da hier extrem viele Möglichkeiten mit verschiedensten Problemen existieren, wird dieser Punkt vorest vernachlässigt.

[8] https://developers.google.com/youtube/v3/
[9] https://developer.apple.com/documentation/musickitjs/
[10] http://developers.deezer.com/login?redirect=/api

3.4 Sonstiges

3.4.1 3D Print version

Da unsere Box nicht übermäßig groß sein soll, müssten handelsübliche 3D-Drucker von der Größe ausreichend sein. Das Modellieren einer 3D-Print Version ist am Ende von der Expertise der Gruppe abhängig. Abgesehen davon sollte es kein besonderes Problem darstellen.

3.4.2 Sound Wiedergabe auf der Box selbst

Manche Pi Modelle verfügen über einen On-Board Audio Anschluss. Die Wiedergabe über diesen ist qualitativ für Musik meist ungeeignet und sollte daher über ein weiteres Audiomodul oder eine externe Soundkarte erfolgen. Innerhalb der Raspberri Pi Reihe gibt es dafür Accessoires, die circa 20-30€ kosten.[11]. Zur Wiedergabe auf der Box selbst müsste dafür auf dem Raspberry eine Spotify Instanz laufen, damit auch diese als Connected Speaker erkannt wird. Hierfür existieren Libraries wie *raspotify*[12].

3.4.3 Box unter 30€ Kosten

Mit einem Raspberri Pi wäre dieses Ziel möglich, es könnte aber kein Pi ab Model 3 verwendet werden, da diese über dem Ziel liegen. Mit dem Raspberri Pi Zero W mit eingebautem W-Lan und einem USB Port für den NFC-Reader gäbe es ein kostengünstiges Model, welches für ca. 10$ erhältlich ist [13].

4 Design Mockups

TODO: setze pdfs ein

5 Durchführung

5.1 Technologien und Hilfsmittel

TODO: Vue, vscode, devops krams, etc Entwickelt wird mit Visual Studio Code, da es eine einfache Nutzung des Linux-Subsystems ermöglicht. [14].

5.2 Deployment Cycle

TODO: ziehe ticket > assigne dich selbst > draft PR > wenn fertig, setzte *undraft* > assigne 2 reviewer > merge master

5.3 Probleme während der Durchführung

TODO:

[11] https://www.raspberrypi.org/products/
[12] https://github.com/dtcooper/raspotify
[13] https://www.raspberrypi.org/products/raspberry-pi-zero-w/
[14] https://code.visualstudio.com/docs/remote/wsl

6 Projekt Meilensteine

6.1 Meilenstein 1

TODO: Kurze Einleitung, von wann bis wann ging

6.1.1 Ziel

TODO: was haben wir uns vorgenommen, was war das ziel, was wollten wir schaffen?

6.1.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.1.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.1.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.1.5 Retrospektive

TODO: Was haben wir dabei gelernt? Neue Erkentnisse? Neue Sichtweisen? Was lief gut, neu gelernt, was lief nicht so gut, was verbessern?

6.2 Meilenstein 3

TODO: Kurze Einleitung, von wann bis wann ging

6.2.1 Ziel

TODO: was haben wir uns vorgenommen, was war das ziel, was wollten wir schaffen?

6.2.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.2.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.2.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.2.5 Retrospektive

TODO: Was haben wir dabei gelernt? Neue Erkentnisse? Neue Sichtweisen? Was lief gut, neu gelernt, was lief nicht so gut, was verbessern?

6.3 Meilenstein 4

TODO: Kurze Einleitung, von, bis

6.3.1 Ziel

TODO: was haben wir uns vorgenommen, was war das ziel, was wollten wir schaffen?

6.3.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.3.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.3.4 Erkenntnisse

Was haben wir dabei gelernt? Neue Erkentnisse? Neue Sichtweisen?

6.3.5 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.3.6 Retrospektive

TODO: Was haben wir dabei gelernt? Neue Erkentnisse? Neue Sichtweisen? Was lief gut, neu gelernt, was lief nicht so gut, was verbessern?

7 Erkenntnisse

8 Code Walkthrough

TODO: vielleicht interessante Code Snippets?

9 Testing

TODO: wie haben wir getestet, haben wir getestet?

10 Technische Diagramme

TODO: ER Diagramme, UML, solcher krams