

# Projektbericht Smart Music Player

Anton Bracke

Jan Eberlein

Tom Calvin Haak

Julian Hahn

Nick Loewecke

27. Februar 2021

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>4</b>
1.1	Unternehmen . . . . .	4
1.2	Anforderungsanalyse . . . . .	4
1.2.1	Produktvision . . . . .	5
1.2.2	Anforderungen . . . . .	5
<b>2</b>	<b>Projektplanung</b>	<b>6</b>
2.1	Projekt Management . . . . .	6
2.2	Vor den Sprints . . . . .	7
2.3	Arbeit in den Sprints . . . . .	7
2.4	Am Ende der Sprints . . . . .	8
2.5	Quelloffenes Arbeiten . . . . .	8
<b>3</b>	<b>Machbarkeitsstudie</b>	<b>9</b>
3.1	NFC Tag . . . . .	9
3.2	Raspberry Pi . . . . .	10
3.3	User Interface . . . . .	11
3.4	Spotify . . . . .	12
3.5	Leek Box / Backend . . . . .	12
3.6	Andere Musikservices . . . . .	13
3.7	Sonstiges . . . . .	13
<b>4</b>	<b>Tools und Vorgehen</b>	<b>14</b>
4.1	Hilfsmittel . . . . .	14
4.1.1	Github & Github IO . . . . .	14
4.1.2	Visual Studio Code & WSL . . . . .	16
4.1.3	Prototyping (Figma) . . . . .	16
4.1.4	Lucidchart . . . . .	17
4.2	Entwicklungszyklus . . . . .	17
4.3	Testing . . . . .	19
<b>5</b>	<b>Technische Umsetzung</b>	<b>21</b>
5.1	Projektstruktur . . . . .	21
5.2	Technologien . . . . .	22
5.2.1	Programmiersprachen . . . . .	22
5.2.2	Frameworks . . . . .	23
5.2.3	Containervirtualisierung (Docker) . . . . .	29
5.2.4	Reverse-Tunnel (ngrok) . . . . .	30
5.2.5	Externe Bibliotheken . . . . .	30

5.3	Hardware . . . . .	31
<b>6</b>	<b>Durchführung</b>	<b>32</b>
6.1	Meilenstein 1 . . . . .	32
6.2	Meilenstein 2 . . . . .	33
6.3	Meilenstein 3 . . . . .	36
6.4	Meilenstein 4 . . . . .	40
6.5	Meilenstein 5 . . . . .	47
6.6	Meilenstein 6 . . . . .	50
6.7	Meilenstein 7 . . . . .	55
<b>7</b>	<b>Fazit</b>	<b>56</b>
7.1	Ausblick . . . . .	56
7.2	Projektreflexion . . . . .	57
<b>8</b>	<b>Anhang</b>	<b>60</b>

# 1 Einleitung

Ein zentraler Teil des Studiengangs „Informationstechnologie“ ist es, Softwareentwicklung in Teams und Kommunikation mit Kund:innen zu erlernen. [Kie21] Dies passiert vor allem im Modul „Projekt Informatik (PROI)“. In diesem arbeiten Studierende ein Semester lang in kleinen Gruppen an verschiedenen Projekten. Diese bilden häufig reale Sachverhalte der Softwareentwicklung ab und werden meist von Firmenpartner:innen aus der Wirtschaft gestellt. Die Teams wählen ein oder mehrere Projekte, die sie gerne bearbeiten würden, und stellen sich mit einer Kurzbewerbung bei den entsprechenden Firmen vor. Welche Teams die jeweiligen Projekte bearbeiten, entscheiden die Firmen selbst. Dieser Bericht beschreibt das Projekt und dessen Verlauf, dass das Autoren-Team für die macio GmbH aus Kiel bearbeitete.

## 1.1 Unternehmen

Das Unternehmen macio stellte in ihrer Ausschreibung als einzige Firma die Anforderung, dass Projekt quelloffen (*Open Source*) umzusetzen. Das Team schätzte die Bereitschaft, die geleistete Arbeitszeit nicht für den Eigenbedarf zu nutzen, sondern einen Beitrag für die Open-Source-Community zu leisten. Damit wurde die folgende Definition für Open-Source erfüllt:

*Ich denke, dass jede allgemein nützliche Information frei sein sollte. Mit 'frei' beziehe ich mich nicht auf den Preis, sondern auf die Freiheit, Informationen zu kopieren und für die eigenen Zwecke anpassen zu können. Wenn Informationen allgemein nützlich sind, wird die Menschheit durch ihre Verbreitung reicher, ganz egal, wer sie weiter gibt und wer sie erhält. [Tor]*

Das Hauptgeschäft von macio ist das Design und Entwickeln von Mensch-Maschine-Interfaces. Als etabliertes Unternehmen konnten Sie neben der Rolle des Kunden auch ihre jahrelange Erfahrung bei Bedarf einbringen und Hilfestellung leisten. So gab beispielsweise ein Designer des Unternehmens hilfreiches Feedback zum Entwurf der Benutzeroberfläche.

## 1.2 Anforderungsanalyse

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. Da die aktuell am Markt erhältlichen Konkurrenzprodukte wenig konfigurierbarkeit bieten, zum Beispiel kein anderer

Lautsprecher ausgewählt werden kann, sollte hier eine Open-Source Alternative geschaffen werden, die dem Nutzer volle Freiheiten zur Anpassung bietet.

### 1.2.1 Produktvision

Mithilfe der von macio gestellten Projektskizze (siehe 8) erstellte das Team die folgende Produktvision: Angestrebt wurde die Konzeption und Entwicklung einer Musik-Box, die NFC Tags lesen kann und Schnittstellen zu Musik-Streaming-Anbietern (wie Spotify) besitzt. Wird auf die Box beispielsweise ein Spielzeuge (z.B. in Form von kleinen Figuren) mit integrierten NFC-Chips gestellt, soll ein frei konfigurierbares Musikstück über eine bereits bestehende Musik-Anlage abgespielt werden. 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 stellte die benötigte Hardware zur Verfügung und unterstützte bei technischen Fragen.

### 1.2.2 Anforderungen

Aus der Projektskizze und der Produktvision wurden folgende konkrete Mindestanforderungen und entsprechende Klassifikationen an das Produkt abgeleitet:

Anforderung	Klassifikation
NFC-Tags lesen, schreiben und entschlüsseln	funktional
Mit Spotify Connect verbinden und arbeiten	funktional
Responsive UI konzeptionieren und umsetzen	funktional
Aussagekräftige Dokumentation mit Benutzerhandbuch	nicht funkt.

Tabelle 1: Mindestanforderungen

Um diese Anforderungen weiter zu definieren erstellte das Team User Stories, durch die sichergestellt wurde, dass beide Parteien (der Kunde und die Entwickler) die Anforderungen gleich interpretierten und verstanden. Dabei entstand auch eine Priorisierung der Anforderungen, bemessen an dem *Return of Investment* und dem Nutzen für den Endnutzer. Im weiteren Gespräch mit den Kunden ergaben sich ergänzende Anforderungen an das Produkt:

Anforderung	Klassifikation
Sound Wiedergabe auf der Box selbst	funktional
Unterstützung anderer Musikdienste	funktional
3D-Modellierung und Print einer passenden Box	nicht funkt.
Maximaler Kostenpunkt von 30€	nicht funkt.
Cloud-Anbindung der Box	funktional
Auslieferung des UI aus der Cloud	funktional

Tabelle 2: ergänzende Anforderungen

Durch die agile Projektorganisation wuchsen diese Anforderungen iterativ von Sprint zu Sprint und neue implizite Anforderungen entstanden. Um die Kosten von maximal 30€ zu realisieren, entstand die Idee zur Konzeption von zwei Versionen der *Leek-Box*. Während die kostengünstige Variante nur die Grundfunktionen umsetzt, bietet die teurere einen größeren Funktionsumfang, wie das Abspielen der Musik über einen eigenen Lautsprecher. Einige der Anforderungen haben sich während der Entwicklungsphase als widersprüchlich herausgestellt. So wurde sich zum Beispiel gegen bezahlte Services entschieden. Dies schränkte die Usability ein, entsprach jedoch mehr dem Open-Source-Gedanken des Projekts. Ob und wie diese Anforderungen umgesetzt werden können wird im späteren Abschnitt Machbarkeitsstudie (Abschnitt 3) behandelt.

## 2 Projektplanung

### 2.1 Projekt Management

Am Anfang des Projekt entschied sich das Team in Abstimmung mit macio für eine agile Projektorganisation. Diese Richtung des Projektmanagements zeichnet sich vor allem durch fortlaufende Produktentwicklung, kontinuierliches Feedback, kooperatives Arbeiten und Reaktionsfähigkeit bei Anforderungsänderungen aus. Diese Entscheidung geschah aus mehreren Gründen. Zum einen haben agile Methoden kein festes Endprodukt als Ziel, wie es beispielsweise bei klassischer Softwareentwicklung (Wasserfallmodell, V-Model, etc) der Fall ist. Solch ein festes Ende würde der geforderten Veröffentlichung als Open-Source-Projekt nicht gerecht werden, da solche per Definition erweiterbar sind. Des Weiteren haben agile Entwicklungsprozesse vergleichsweise kürzere Veröffentlichungszyklen, was dem Team ermöglichte, ein funktionierendes Produkt innerhalb des recht knappen Zeitrahmens eines Semesters<sup>1</sup> zu erstellen.

<sup>1</sup>Dieses Semester war aufgrund besonderer CoViD19-bedingten Auflagen einige Wochen kürzer als ein typisches.

Entsprechend der Entscheidung zu agiler Entwicklung wurde zu Beginn des Projekt keine feste Planung des Projektverlaufs aufgestellt. Stattdessen entwickelte das Team eine Methodik um iterativ am Projekt zu arbeiten, wobei sich an der Methode Scrum orientiert wurde. Aus dieser wurden vor allem die Einteilung in Projektiterationen fester Länge (Sprints) und die zugehörigen wiederkehrenden Meetings übernommen. Diese Sprintlänge betrug im allgemeinen zwei Wochen<sup>2</sup>. Die Rollen der Teammitglieder in Scrum wurde nicht adaptiert, da sich im Team für eine Gleichverteilung der Aufgaben entschieden wurde.

## 2.2 Vor den Sprints

Den Anfang eines jeden Sprints stellte das Sprint-Planing dar. Im diesem stellte das Team eine Vision für die Sprintiteration auf. Diese wurde gemeinsam mit dem Kunden abgesprochen und gegebenenfalls abgeändert. Im Anschluss wurde die Produktvision vom Team in technisch orientierte Arbeitspakete aufgeteilt. Dies geschah anhand des geschätzten zeitlichen Arbeitsaufwands der einzelnen Änderungen. Falls sich hierbei gegebenenfalls herausstellte, dass einzelne Arbeitspakete nicht umsetzbar oder zu aufwändig waren, wurde die Produktvision entsprechend angepasst.

## 2.3 Arbeit in den Sprints

Während der Sprints wurde die Produktvision aus den Sprint-Planings implementiert. Alle Mitglieder des Team waren an dieser Arbeit beteiligt. Die einzelnen Arbeitspakete wurden entweder alleine, in Paaren oder selten auch in Gruppen bearbeitet. Dies wurde anhand der Anforderung und des Aufwands der jeweiligen Arbeitspakete entschieden. Auch das spezifische Fachwissen der beteiligten Personen hatte einen Einfluss auf die Aufteilung. In jedem Fall wurden die Pakete erst bei Arbeitsbeginn und nur von den bearbeitenden Mitgliedern selbst zugewiesen.

Nach Bearbeitung eines Paketes wurden die vollzogenen Änderungen durch Code Reviews von mindestens zwei anderen Entwicklern geprüft. Nur wenn diese erfolgreich waren, wurde der entsprechende Code ins Produktinkrement übernommen. Dieses Verfahren wurde angesetzt, um sowohl Flexibilität von Arbeitszeiten zu ermöglichen, als auch um die gewünschte Produktqualität sicherzustellen. Verschiedene Arbeitszeiten waren notwendig, da alle Teammitglieder verschiedene parallele Veranstaltungen besuchten und anderen beruflichen Tätigkeiten nachgingen. So waren keine langfristigen synchronen Arbeitszeiten aller Mitglieder möglich. Die freie Verteilung von Paketen ermöglichte auch, dass

---

<sup>2</sup>Abgewichen wurde von dieser Länge nur zum Jahresende, um die Feiertage unterzubringen.

Mitglieder in persönlich bevorzugten Themengebieten arbeiten konnten. Dies half die Motivation des Teams am Projekt hochzuhalten.

Um sich während der Sprints abzustimmen und um Arbeitsfortschritte abzugleichen, trafen sich die Teammitglieder drei Mal pro Woche. Dies geschah immer montags, mittwochs und freitags. Bei Bedarf wurde auch von dieser Taktung abgewichen. In Aufbau und Zweck orientierten sich diese regelmäßigen Meetings an den „Daily Standups“ der Scrum-Methode. Während den Meetings stellte jedes Teammitglied kurz vor was es seit dem letzten Standup am Projekt bearbeitet hatte, welche Probleme dabei aufgetreten waren und was bei der Arbeit gelernt wurde. Auch welche Themen jedes Mitglied bis zum nächsten Standup bearbeiten wollte, wurde besprochen. Nach diesen Kurzvorstellungen wurden einzelne besonders interessante und wichtige Punkte der Arbeit oder gelöste Probleme im Detail besprochen. So konnte das Team auf den gleichen Wissensstand kommen und gemeinsam Entscheidungen treffen.

## 2.4 Am Ende der Sprints

Nach jedem Sprint wurde das Ergebnis bzw. das Produktinkrement dem Kunden und der Projektbetreuung vorgestellt. Hier wurde meist auch zusammen mit macio das Ausmaß des nächsten Produktinkrements besprochen. Die resultierenden Wünsche und Rückmeldungen wurden mit ins Backlog und die folgenden Besprechungen mitgenommen. Im Anschluss fand immer eine Retrospektive statt. In dieser besprach das Team die eigene Zusammenarbeit und den gemeinsamen Umgang. Hierfür stellten sich alle Teammitglieder für sich folgende Fragen für die Arbeit am entsprechenden Sprint:

- Was lief gut?
- Was lief schlecht?
- Was habe ich neu gelernt?
- Was können wir in Zukunft besser machen?

Die jeweiligen Antworten wurden zusammengetragen und im Team gemeinsam reflektiert. Das Feedback in diesem Kontext war konstruktiv und fair aber auch ehrlich. Die daraus entstandenen Erkenntnisse und Verbesserungsvorschläge wurden genutzt um das Teamwork und die Arbeit im folgenden Sprint weiter zu optimieren. Direkt nach jeder Retrospektive startete der nächste Sprint beginnend mit dem entsprechenden Sprint-Planing.

## 2.5 Quelloffenes Arbeiten

Teil der Anforderung war, dass die Software als Open-Source-Projekt der Öffentlichkeit zur Verfügung stehen soll. Dementsprechend entwickelte das Team den Anspruch, das dieses Projekt nicht nur Open-Source sondern auch erweiterbar



und verständlich sein soll. Aufgründessen wurde die Entscheidung getroffen, das Projekt von Entwicklungsbeginn öffentlich auf GitHub aufzusetzen. Diese Plattform bot sich an, da sie für den Umfang des Projekts viele hilfreiche Funktionalitäten in den Bereichen Automatisierung und Kollaboration bietet, allerdings trotzdem kostenlos ist. Diese Werkzeuge halfen dabei den Entwicklungsprozess sehr intuitiv und zentral zu gestalten. So wurden zum Beispiel die meisten Diskussionen zu fachlichen Themen direkt in den Arbeitspaketen. Auch im Nachhinein können dann Entscheidungen und Denkprozesse von anderen Kollaborierenden nachvollzogen werden. Dies hilft zusätzlich bei der zukünftigen Instandhaltung und Weiterentwicklung des Projekts.

Das Produkt-Backlog wurde auch in dieses GitHub Repository eingepflegt. Dies erhöhte einerseits die Übersichtlichkeit über den Stand des Projekts innerhalb, da Arbeitspakete und zugehöriger Code sehr einfach verknüpft werden konnten. Andererseits ermöglicht diese Zusammenlegung auch, dass andere Entwickler:innen einfach ins Projekt einsteigen und ihre Arbeit dokumentieren können, ohne andere Plattformen aufsuchen zu müssen. Die Möglichkeiten der Automatisierung wurden genutzt, um den Workflow zu vereinfachen und um die Code-Qualität sicherzustellen. Dies geschah zum Beispiel die durch die automatische Erstellung von Branches für einzelne Arbeitspakete bei Beginn der Bearbeitung und automatische statische und dynamische Tests des Codes.

Darüber hinaus war die Open-Source Anforderung auch hilfreich bei der Produktentwicklung. So konnte die Zielgruppe für dieses Projekt auf Menschen mit fortgeschrittenen Technikenntnissen und Interesse an Bastelprojekten eingeschränkt werden. Dies war vor allem für das erstmalige Einrichten und die zugehörige Dokumentation maßgebend. Entsprechend konnte sich hier eher technischer Sprache bedient werden. Allerdings galt diese Einschränkung der Zielgruppe auch nicht für alle Bereiche des Projekts. Die Benutzbarkeit der Nutzoberfläche sollte einfach und ohne Vorkenntnisse möglich sein, da die Musikbox z. B. verschenkt oder in Familien genutzt werden könnte. Dieser Anwendungsfall schließt andere Nutzende ein, als bei der Einrichtung.

## 3 Machbarkeitsstudie

### 3.1 NFC Tag

#### lesen und schreiben

Zur Nutzung von NFC Tags müssen diese gelesen und deren Daten an das Backend übertragen werden können. Hierfür ist ein Hardware NFC-Reader notwendig, der die Daten ausliest und an den Raspberry Pi kommuniziert. Dafür werden Tastatureingaben, die die Tag-IDs darstellen, emuliert. *Evdev*, ein Kernel

Modul von Linux, könnte dann zum Abgreifen dieser Eingaben genutzt werden. Mit *node-evdev*<sup>3</sup> kann *evdev* auch mit Node genutzt werden. Um dieses Modul auch für den Raspberry Pi und mit Typescript nutzbar zu machen, müssten noch Änderungen an diesem Modul durchgeführt werden.

Um neben der initial gespeicherten ID weitere Daten auf einen NFC-Tag schreiben zu können, benötigt der NFC-Tag einen eigenen Speicher. Ist dieser vorhanden, können dort z. B. User Daten hinterlegt werden. Dafür wäre ein spezieller NFC-Reader, mit Schreibfunktionalität notwendig.

## 3.2 Raspberry Pi

### Docker Integration

Docker Integration sollte für einfachere Auslieferbarkeit angeboten werden (vgl. 5.2.3). Die Installation von Docker auf einem Raspberry Pi wird in vielen Anleitungen online beschrieben<sup>4</sup>, weshalb die Umsetzbarkeit dieses Punktes als sehr wahrscheinlich eingeschätzt wird.

### Öffentlich zugängliches Web Interface

Zur Bereitstellung der Benutzeroberfläche für eine Box soll der Raspberry Pi verwendet werden. Um die Erreichbarkeit dieser Web Anwendung auch von außerhalb des eigenen Netzwerkes zu ermöglichen, könnte eine Portweiterleitung im Router eingerichtet werden. Dies ermöglicht den Zugriff auf die Benutzeroberfläche durch die öffentliche IP-Adresse des Routers. Da die öffentliche IP-Adresse eines privaten Internet-Anschlusses häufig nicht konstant ist, könnte zum Zugriff ein DynDNS Dienst verwendet werden, welcher eine feste Domain in die wechselnde IP Adresse des Routers übersetzt. Zur Umsetzung dieses Punktes mittels DynDNS könnten die in diesem Feld unerfahreneren Entwickler etwas Einarbeitungszeit benötigen, was die Abschätzung des Arbeitsaufwandes sehr abhängig von den Erfahrungen macht. Alternativ könnte für das Projekt auch *Github.io Pages* genutzt werden, was impliziert, dass jede *Leek-Box* kein eigenes Frontend hostet, sondern dieses auf einem von GitHub bereitgestellten Web-Server öffentlich erreichbar wäre. In diesem Fall müsste die Kommunikation zwischen dem öffentlichen Web-Server und dem Backend auf dem Raspberry Pi implementiert werden.

---

<sup>3</sup><https://github.com/sdumetz/node-evdev>

<sup>4</sup><https://phoenixnap.com/kb/docker-on-raspberry-pi>

### 3.3 User Interface

#### Zugriff auf NFC Reader von Cloud Anwendung

Zum Zugriff von Frontend zu Backend könnte eine REST-API verwendet werden, sodass Daten durch API-Calls ausgetauscht werden könnten.

#### Login via Spotify, Youtube, etc.

Zur Feststellung, ob ein Benutzer bei einem Musik-Service angemeldet ist, muss sich dieser authentifizieren. Dabei muss in den meisten Fällen auch das Abomodell ermittelt werden, um die Möglichkeit zur Nutzung von bestimmten Funktionen, wie der freien Wahl eines Songs, festzustellen. Dafür soll das offene Protokoll *OAuth* verwendet werden, welches von den häufig genutzten Musik-Streaming-Anbietern (Spotify, Deezer, Amazon-Music) unterstützt wird. Es existieren mehrere Bibliotheken, welche die Nutzung von OAuth Provider abstrahieren, was zu einer verminderten Komplexität führt und somit die Umsetzung erleichtert.

#### Nutzer verschiedener Streaminganbieter wiedererkennen

Sollte ein Nutzer mehrere Musikstreaminganbieter verwenden wollen, muss eine eindeutige Identifikation anhand eines Merkmals sichergestellt werden. Die E-Mail wäre hierbei ein geeignetes Merkmal zur Wiedererkennung des Benutzers, da diese einzigartig ist. Über gesetzte Scopes in der OAuth Anfrage kann diese vom jeweiligen Provider mitgeliefert werden. Dafür müsste zum Beispiel vorausgesetzt werden, dass ein Nutzer die gleiche E-Mail bei den Musikstreamingdiensten nutzt. Da dies jedoch nicht der Realität entspricht, könnte dem (bereits eingeloggten) Nutzer die Möglichkeit gegeben werden, weitere Accounts zu dem bestehenden hinzuzufügen und entsprechend in der Datenbank zu hinterlegen.

#### Musik Artwork laden

Zur Wiedererkennung eines Musikstücks könnte das zugehörige Artwork geladen und gespeichert werden. Beim primär verwendeten Musikservice Spotify sollte dies kein Problem darstellen, da zu jeder Anfrage von Titeln oder Liedern auch das Albumcover mitgeliefert wird.<sup>5</sup>

#### Eigene Bilder nutzen

Um den Nutzer weitere Personalisierung seiner angelegten Tags zu ermöglichen, könnten auch eigene Bilder für NFC-Tags verwendet werden. Eigene Bilder hochzuladen sollte generell möglich sein. Der einfachere Weg hierbei wäre es, dass

---

<sup>5</sup><https://stackoverflow.com/questions/10123804/retrieve-cover-artwork-using-spotify-api>

Nutzer ein externes Bild verlinken, dies würde ohne größerer Aufwand möglich sein. Sollte der Nutzer eigene Bilder hochladen wollen, würde dies mehr Aufwand beanspruchen, da die Daten zum Backend gesendet und dort gespeichert werden müssten. In unserem Kontext mit Vue.js und Node.js würde das Plugin *vue-picture-input* helfen. Mit einem *Axios Post* könnte so das Bild an das Backend gesendet werden. <sup>6</sup>.

### 3.4 Spotify

#### Spotify Connect Lautsprecher auswählen und speichern

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*. <sup>7</sup> 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.

### 3.5 Leek Box / Backend

#### Leek-Box mit Cloud-Anwendung 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.

#### 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.

---

<sup>6</sup><https://www.digitalocean.com/community/tutorials/vuejs-uploading-vue-picture-input>

<sup>7</sup><https://developer.spotify.com/documentation/web-api/guides/using-connect-web-api/>

## 3.6 Andere Musicservices

### Unterstützung von Youtube Music

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

### Unterstützung von Youtube

Youtube bietet die Möglichkeit, nach Videos zu suchen<sup>8</sup>. 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.

### Unterstützung von Apple Music

Apple Music bietet hier mit deren MusicKit JS<sup>9</sup> eine Möglichkeit, um Musik abzuspielen.

### Unterstützung von Deezer

Deezer lässt sich vergleichbar zu Spotify über eine API steuern.<sup>10</sup>

## 3.7 Sonstiges

### 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.

### Sound Wiedergabe auf internem Lautsprecher

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 Raspberry Pi Reihe gibt es dafür Accessoires, die circa 20-30€ kosten.<sup>11</sup>. 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*<sup>12</sup>.

---

<sup>8</sup><https://developers.google.com/youtube/v3/>

<sup>9</sup><https://developer.apple.com/documentation/musickitjs/>

<sup>10</sup><http://developers.deezer.com/login?redirect=/api>

<sup>11</sup><https://www.raspberrypi.org/products/>

<sup>12</sup><https://github.com/dtcooper/raspotify>

### Kostenpunkt unter 30€

Mit einem Raspberry Pi wäre dieses Ziel möglich, es könnte aber kein Pi ab Model 3 verwendet werden, da diese über dem Ziel liegen. Mit dem Raspberry 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 <sup>13</sup>.

## 4 Tools und Vorgehen

### 4.1 Hilfsmittel

Im Rahmen des Projekts nutze das Team eine Vielzahl von Hilfsmitteln, die die Zusammenarbeit und Produktivität des Teams steigern sollte. Diese wurden zum einen zu Beginn des Projektes in einer Gruppendiskussion herausgearbeitet oder als Ergebnis der Retrospektiven erprobt.

#### 4.1.1 Github & Github IO

Da alle Teammitglieder, wie auch die Kunden bereits einen Account bei dem Dienst *GitHub* besaßen, wurde die Entscheidung getroffen, diesen als Host für *Git-Remote-Repositories* einzusetzen. Aufgrund der bereits gewonnenen Erfahrung mit diesem Dienst und der Fokussierung auf quelloffene Software, stellte er für dieses Projekt die ideale Umgebung zur Quellcodeverwaltung und Kollaboration dar.

### Kollaboration

Mit dem Anspruch das Produkt auch nach Projektende fortzuführen, wurden auch die Projektmanagement Funktionen von GitHub verwendet. Die einzelnen Aufgaben bzw. Arbeitspakete (*Issues*<sup>14</sup>) wurden in einem *Issue Board* den verschiedenen Arbeitsstatus<sup>15</sup> zugeordnet. Außerdem konnten verschiedene Informationen, wie z. B. der Sprint bzw. *Milestone* <sup>16</sup> in dem das Ticket abgeschlossen sein soll oder der bearbeitende Entwickler dokumentiert werden. Da diese Ansicht ebenfalls nach diversen Eigenschaften gefiltert werden kann, konnte sich jeder Entwickler schnell einen Überblick über den aktuellen Projektstand verschaffen. Das Team setzte zur Unterstützung außerdem den *Renovate-Bot*<sup>17</sup> ein, welcher die Tickets automatisch anhand von verschiedensten Events verschob und einen Branch anlegt, wenn sich ein Entwickler einem Ticket zuordnet.

---

<sup>13</sup><https://www.raspberrypi.org/products/raspberry-pi-zero-w/>

<sup>14</sup>Da das Team größtenteils den Begriff *Issue* nutzte, wird dieser auch fortlaufend hauptsächlich verwendet.

<sup>15</sup>Status = Backlog, Todo, In Progress, Review und Done

<sup>16</sup>Auch hier wird im weiteren Verlauf eher der Begriff Milestone oder Meilenstein verwendet.

<sup>17</sup><https://github.com/renovatebot/renovate>

Eine weitere Funktionalität von GitHub, die in diesem Projekt genutzt wurden sind die sogenannten *Actions*<sup>18</sup>. Sie ermöglichen automatisierte Tests, die den Quellcode und die Anwendung als Gesamtkonstrukt auf verschiedenste Fehler prüfen und frei konfigurierbar sind. In diesem Projekt wurden die folgenden vier Tests durchgeführt um die Änderungen vor der Übertragung in den Master-Branch zu validieren:

1. **build**: Wird die Software fehlerfrei gebaut?
2. **lint**<sup>19</sup>: Ist der Quellcode in gutem Stil geschrieben?
3. **test**: Sind die geschriebenen Unit-Tests erfolgreich?
4. **typecheck**: Sind alle Typen kompatibel?

Sind alle Tests erfolgreich, ist die Änderung technisch nutzbar.

Außerdem wurde die in GitHub existierende Funktionalität der Pull Requests verwendet. Beim *Pushen*<sup>20</sup> werden die Änderungen automatisch in dem den Branch betreffenden Pull Request angezeigt (wenn vorhanden). Jeder Pull Request enthält allgemeine Informationen, welchem Zweck er dient und von wem die Änderung durchgeführt wird. Außerdem wird das Ergebnis der automatisierten Tests (*GitHub Actions*) angezeigt. Das Projekt ist so konfiguriert, dass die Übernahme der Änderung des Pull Requests nur möglich ist, wenn mindestens zwei andere Entwickler diesen Änderungen durch Code Reviews zugestimmt haben und alle automatisierten Test erfolgreich abgeschlossen wurden. Ist dies der Fall, kann die Änderung in den Master-Stand übertragen werden. Für die Code-Reviews wurde ebenfalls die GitHub interne Funktionalität genutzt, die bequeme Änderungsvorschläge ermöglicht.

### Github Pages

Zum Bereitstellen des Frontends der *leek-boxen* wird der Dienst GitHub Pages verwendet. Dieser stellt pro Organisation und Repository eine kostenlose URL `xyz.github.io` zur Verfügung. Dabei werden unter dieser statische Dateien, wie zum Beispiel die einer Website: *.html*, *.css*, *.js*, aus einem bestimmten Branch oder einem Ordner bereitgestellt. Die Konfigurationsanforderungen für ein einfaches Website-Hosting sind somit minimal.

---

<sup>18</sup><https://github.com/features/actions>

<sup>19</sup>Tool zur statischen Codeanalyse

<sup>20</sup>Hochladen der Änderung vom Entwickler-PC auf das Repository auf GitHub

### 4.1.2 Visual Studio Code & WSL

Als Entwicklungsumgebung in diesem Projekt wurde *Visual Studio Code* verwendet. VSCode war allen Gruppenmitgliedern bereits bekannt und wurde von vielen auch regelmäßig verwendet. Die IDE beherrscht den Umgang mit allen für das Projekt benötigten Dateitypen und ist durch eine Vielzahl an kostenlos angebotenen Erweiterungen (Extensions) sehr anpassbar an die Projektbedürfnisse. So wurden in diesem Fall unter anderem die Erweiterungen *Vetur* (für Vue.js), *LaTeX Workshop* (für die Erstellung dieses Berichts) und *Tailwind CSS IntelliSense* (Für Auto-Vervollständigung der CSS Klassen von Tailwind) verwendet. Außerdem bietet *Visual Studio Code* eine komfortable Anbindung<sup>21</sup> an das *Windows Subsystem für Linux (WSL)*<sup>22</sup>, welches eingesetzt wurde, um eine möglichst homogene Arbeitsumgebung innerhalb des Teams sicherzustellen und neben der gleichen IDE auch auf Betriebssystemen der gleichen Kernel-Infrastruktur (UNIX) zu arbeiten. Dies erleichterte insbesondere die Arbeit mit *Docker* deutlich.

### 4.1.3 Prototyping (Figma)

Zur Abstimmung auf ein Design für die Benutzeroberfläche wurden vor der Entwicklung Prototypen der benötigten Komponenten mittels des web-basierten Prototyping-Tools *figma*<sup>23</sup> konzipiert. So konnten neben dem Design auch Abläufe in Form von „Click-Dummys“ erstellt werden und mit dem Kunden abgestimmt werden. *Figma* wurde verwendet, da es einen kollaborativen Zugriff ermöglicht, kostenlos ist und bereits Erfahrung mit der Plattform bestand.

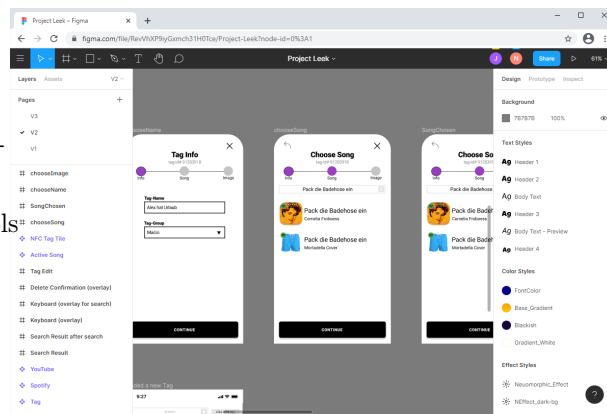


Abbildung 1: Prototypisierung mit *figma*

<sup>21</sup><https://code.visualstudio.com/docs/remote/wsl>

<sup>22</sup><https://docs.microsoft.com/de-de/windows/wsl/about>

<sup>23</sup><https://www.figma.com/>



#### 4.1.4 Lucidchart

Um Abläufe schon vor dem Prototyping skizziert darstellen zu können wurde außerdem die webbasierte Plattform *Lucidchart*<sup>24</sup> verwendet. Hier wurde neben der initial erstellten Produktübersicht auch mehrere Aktivitätsdiagramme erstellt, die aus vom Projektteam entwickelt und mit dem Kunden abgestimmten User-Stories entstanden sind.

Diese sind im Anhang unter 8 zu finden. Auch diese Plattform ermöglicht den gleichzeitigen Zugriff und war den Entwicklern bereits bekannt, was die Notwendigkeit der Einarbeitung eliminierte.

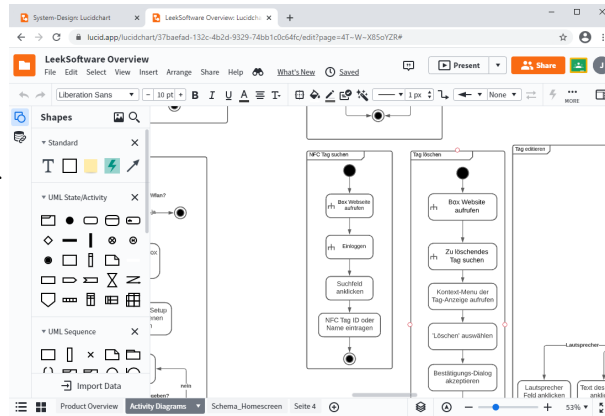


Abbildung 2: Zeichnen von Diagrammen mit *Lucidchart*

## 4.2 Entwicklungszyklus

Während der Umsetzung der *leek-box* durchlief jedes Teammitglied für jedes *Issue* den am Anfang des Projekts festgelegten und durch die Retrospektiven optimierten Zyklus.

Am Anfang steht die Auswahl eines *Issues* aus der Spalte *ToDo* des *Issue Boards*. Nach Identifikation weist sich der Entwickelnde dem Ticket zu. Daraufhin erstellt der verwendete Bot automatisch einen *Branch* in dem Schema *[Ticketnummer]-[Ticketname]*. Nun kann mit der Arbeit in *Visual Studio Code* begonnen werden.

<sup>24</sup><https://www.lucidchart.com/>

Die Änderungen sollten möglichst kleinschrittig aber sinnvoll *comittet* und anschließend *gepusht* werden. GitHub erkennt automatisch die neuen *commits* und erfragt, ob ein neuer *Pull-Request* angelegt werden soll. Dieser hat initial den Status *Open*. Wenn die Aufgaben des *Issues* noch nicht abgeschlossen sind, wird der Status auf *Draft* gesetzt.

Nach dem *push* aller Änderungen, wird das Ergebnis der automatisierten Tests (*GitHub Actions*) ermittelt. Treten hierbei Fehler auf, werden diese geprüft und behoben. Sind alle Tests erfolgreich, betätigt der Entwickelnde die Schaltfläche *Ready for Review*. Wenn ein:e Entwickler:in, diesen *Pull-Request* sieht, wird ein *Code-Review* durchgeführt. Hierzu wird der Branch in die IDE geladen und die Funktionalität verifiziert. Anschließend wird die Code-Qualität geprüft. Dafür werden die geänderten Dateien im Pull-Request betrachtet. Anmerkungen können in Form von Kommentaren an einzelne oder mehrere Zeilen angefügt werden. Bei Änderungsvorschlägen können diese über eine *suggestion* getätigt werden. Ist der komplette Code geprüft wird entschieden, ob die Änderungen angenommen (*approve*) werden oder eine Än-

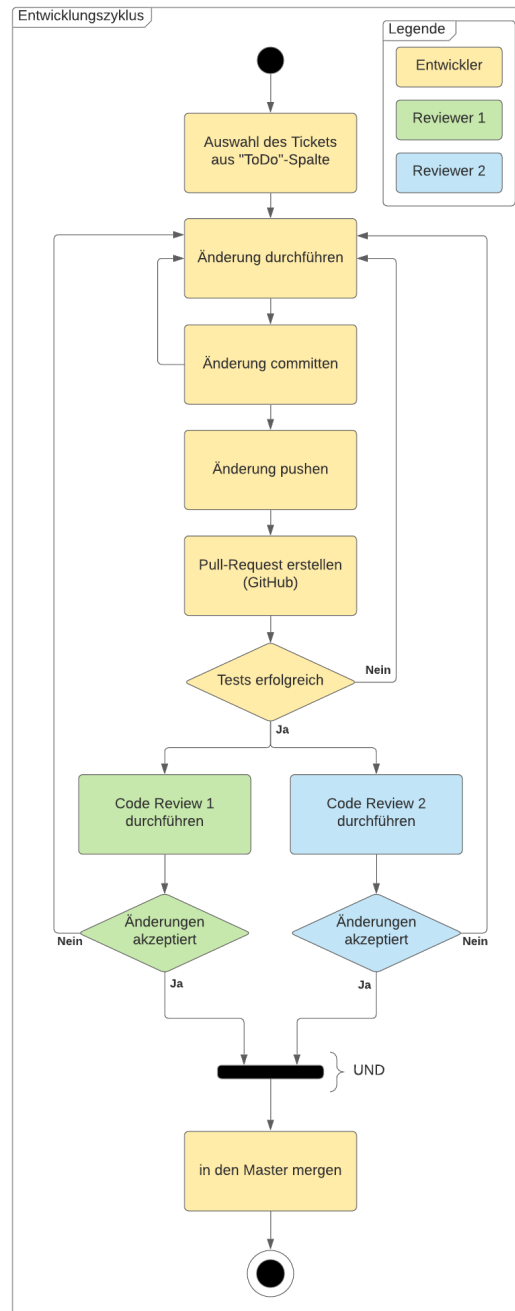


Abbildung 3: Entwicklungszyklus

derungsanforderung gestellt wird (*Request changes*).

Im Falle von Änderungsanforderungen werden diese umgesetzt und ein erneutes Review angefordert. Dies geschieht zyklisch, bis die Änderungen angenommen werden. Sind die Änderungen durch zwei *Code-Reviews* bestätigt worden, werden sie mit der Schaltfläche *Merge into master* übertragen.

### 4.3 Testing

Das Team entschied sich zu Beginn der Produkt-Entwicklung dafür, Tests zur Verbesserung der Codequalität durchzuführen. Quellcode Tests lassen sich in statische und dynamische Tests unterteilen.

#### Statische Tests

Im Allgemeinen untersuchen statische Tests nur Textdokumente und betrachten im Gegensatz zu dynamischen Tests nicht das Verhalten zur Laufzeit. Dies ermöglicht eine häufige und kontinuierliche Nutzung von Tests dieser Art. Zwei der prominenten Varianten statischer Tests, Linting und Reviews, wurden in diesem Projekt genutzt.

#### Linting

Beim Linting wird der Code auf syntaktische Korrektheit geprüft. Auch manche semantische (laufzeitunabhängige) Aspekte werden untersucht. So können kleinere Denkfehler und Flüchtigkeitsfehler, wie z. B. fehlende Kommata, schnell gefunden und behoben werden. Hierfür wurde das Analyse-Werkzeug *ESLint*<sup>25</sup> genutzt. Für diese Werkzeug war zusätzlich auch eine Erweiterung<sup>26</sup> für die Entwicklungsumgebung verfügbar. Mit dieser konnten die Ergebnisse des Linting direkt im Code angezeigt werden. Dies machte den Arbeitsprozess wesentlich zeiteffizienter. Zusätzlich wurde auch die Erweiterung *Vetur*<sup>27</sup> genutzt. Diese stellte die selben Funktionalitäten für den Vue-Framework spezifischen Code zur Verfügung.

Darüber hinaus boten diese Erweiterungen auch Formatierungshilfen mit übertragbaren Konfigurationen, durch deren Nutzung ein konsistenter Code-Stil innerhalb des Teams ermöglicht wurde. Dies legte den Grundstein für lesbaren Code und ermöglichte successive effizientes Arbeiten.

#### Reviews

Teil des Deployment Cycles waren auch Reviews der aktiven Änderungen. Bei diesen wurde der neue bzw. geänderte Quellcode von mindestens zwei anderen

---

<sup>25</sup><https://eslint.org/>

<sup>26</sup><https://marketplace.visualstudio.com/items?itemName=dbaumer.vscode-eslint>

<sup>27</sup><https://vuejs.github.io/vetur/>

Teammitgliedern überprüft. Untersucht wurden vor allem semantische Fehler, Lesbarkeit, Wartbarkeit und Vollständigkeit. Hierfür gab es keinen formalen Plan, allerdings war die Review-Oberfläche von GitHub sehr hilfreich. In dieser wurde eine Änderungsansicht (*diff*) des zu überprüfenden Codes angezeigt, auf der alle Teile des neuen Codes auf einen Blick einsehbar waren. Darüber hinaus bot die Review-Oberfläche auch die Möglichkeit, direkt im Code einzelne oder mehrere Zeilen zu kommentieren. So konnten auch Änderungsvorschläge gemacht und direkt übernommen werden, was für einen effizienten Review-Prozess sorgte.

### Dynamische Tests

Als dynamische Tests wurden in diesem Projekt Anwendungsfall-basierte Tests genutzt. Bei diesen Testfällen werden User-Stories Schritt für Schritt „durchgespielt“ und dabei überprüft, ob das Resultat dem erwarteten Verhalten entspricht. Die meisten Test dieser Art wurden als Integrations-Tests durchgeführt, bei denen die Änderungen im Kontext der bestehenden Software getestet wurden. So konnte vor allem das Zusammenspiel von Frontend und NFC-reader mit dem Backend gut überprüft werden. Aufgrund der Komplexität dieser Kontrollen wurden diese manuell durchgeführt und nicht automatisiert. Die testenden Entwickler durchliefen bei der Überprüfung folgende Schritte:

1. Zu überprüfenden *feature-branch* auschecken
2. Projekt bauen und ausführen
3. Neue oder geänderte Funktionen mit beispielhaften Eingaben nutzen (z.B. neue NFC-Tags anlegen)
4. Gegebenenfalls Probleme und Fehler aufzeichnen und beheben

Diese Tests waren Teil des Deployment Cycles und wurden so in den meisten Fällen parallel zu den Reviews durchgeführt.

Ein gewisser Teil der anwendungsfallbasierten Test fand auf der Ebene von Code-Funktionen (*Units*) statt, welche zur einfacheren Wiederholbarkeit automatisiert wurden. Dazu wurde das JavaScript-Test-Framework *Jest*<sup>28</sup> genutzt, da sich Tests hiermit sehr intuitiv und ohne großen Lernaufwand schreiben ließen. Auch diese Tests waren Teil des Deployment Cycles und wurden so bei jedem Pull-Request automatisch ausgeführt.

---

<sup>28</sup><https://jestjs.io/>

## 5 Technische Umsetzung

### 5.1 Projektstruktur

Die an das Projekt gesetzten Anforderungen machten die Erstellung mehrerer Applikationen notwendig. Neben einer Benutzeroberfläche, auf der die NFC-Tags verwaltet werden können, musste die Steuerung des NFC-Readers und ein System zur Speicherung der Daten sowie zum Abspielen der Musik entwickelt werden. Die dafür benötigten Anwendungen wurden als Microservices konzipiert und umgesetzt. Microservices sind kleine, separat ausführbare und eigenständige Dienste mit geringem Umfang. Diese kommunizieren untereinander mit allgemein verwendeten Protokollen, wie zum Beispiel HTTP, via APIs. Größere Projekte werden bei einer Mikroservice-Architektur dementsprechend in kleine Teile „zerlegt“, die eigenständig entwickelt und betrieben werden können. [Che18]

Zur Verringerung des Aufwands der Konfiguration, des Abhängigkeiten-Managements und der Wartung entschied sich das Team, die einzelnen Microservices auf GitHub in einem Repository zusammenzufassen. [Luc17] Repositories dieser Art werden *mono-repositories* oder *monorepos* genannt, da sich alle Komponenten an einem einzigen Ort befinden. Jeder Service wurde in einem Unterordner in *packages* angelegt (vgl. Abbildung 5.1). Der Ordner *app* enthält die Benutzeroberfläche, *backend* die Datenbankschnittstelle sowie Logik zur Kommunikation mit dem Musik-Streaming-Anbieter und *nfc-reader* die Applikation zum Steuern und auslesen des am Raspberry Pi angeschlossenen NFC-Readers. Im Ordner *commons* befinden sich von allen Services gemeinsam genutzte Ressourcen (wie z. B. Modell-Klassen). Neben den einzelnen Microservices wurden ebenfalls die Dokumentationsdateien wie Setup-Guide, Benutzerdokumentation und Projektbericht im Mono-Repo im Ordner *docs* abgelegt. Auch die für den Dienst *Docker* benötigten Dateien fanden in dem gleichnamigen Ordner Platz. Die Konfigurationsdateien für *ESLint* und den *Typescript*-Compiler sind im Ordner *config* zu finden. Auf die Auflistung aller weiteren Dateien wird in diesem Bericht verzichtet. Eine genaue Aufstellung ist dem *GitHub-Repository*<sup>29</sup> zu entnehmen.

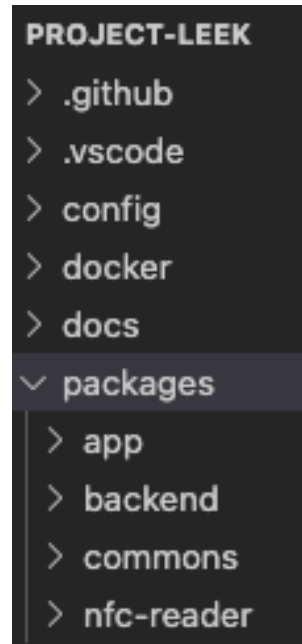


Abbildung 4:  
Orderstruktur

<sup>29</sup><https://github.com/project-leek/project-leek>

Durch die Zusammenfassung in ein Mono-Repository waren die einzelnen Services für jeden Entwickler jederzeit einfach erreichbar, ohne die Notwendigkeit das Repository zu wechseln.

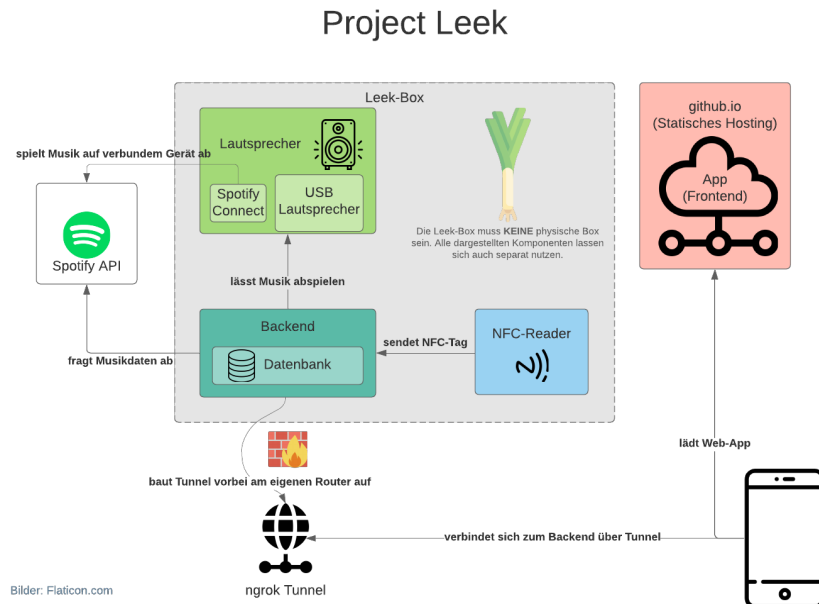


Abbildung 5: Überblick zum Aufbau der Leek-Software

## 5.2 Technologien

### 5.2.1 Programmiersprachen

#### TypeScript

Da alle Teammitglieder bereits Erfahrungen mit JavaScript gesammelt hatten wurde diese zuerst als Programmiersprache vorgeschlagen. Aufgrund von nicht vorhandener Typisierung wurde sich dann jedoch auf die von Microsoft entwickelte Sprache TypeScript geeinigt. Diese bietet aufgrund des Aufbaus auf den ECMAScript-6-Standards eine große syntaktische Ähnlichkeit zu JavaScript. Somit fiel das Erlernen der Sprache den Entwicklern relativ leicht. Durch die starke Typisierung entstehen außerdem weniger Laufzeitfehler, da die entwickelnde Person bereits zur Compile-Zeit auf Typfehler aufmerksam gemacht wird.[Ulb17] Bei der Kompilierung von TypeScript-Code wird dieser zu gültigem JavaScript Code umgewandelt, wodurch auf eine Vielzahl von bereits existierenden JavaScript-Paketen zugegriffen werden konnte. Außerdem lässt sich TypeScript durch Nutzung der Laufzeitumgebung Node.js auch als Backendsprache

verwenden und ersparte dem Team damit die Nutzung von verschiedenen Programmiersprachen.

### 5.2.2 Frameworks

Neben der Programmiersprache Typescript setzte das Team verschiedene Frameworks ein, um mit weniger Aufwand entwickeln zu können.

#### Vue.js

Zur Gestaltung der Benutzeroberfläche für die *leek-box*, die zum Verwalten der NFC-Tags und Steuern der Ausgabegeräte genutzt wird, einigte sich das Team auf die Nutzung des clientseitigen JavaScript-Webframeworks *Vue.js*<sup>30</sup>. Dieses wurde vor allem aufgrund dessen Haupteigenschaften *components* und *reactivity* entschieden.

#### Components

Durch Vue lassen sich Webseiten mit HTML, CSS und JavaScript bzw. TypeScript erstellen. Dabei bietet das Framework, im Vergleich zur Webentwicklung ohne Framework, die Möglichkeit sogenannte Single-File-Components (*.vue* Dateien) zu nutzen. In diesen können sich ein Template für das HTML-Grundgerüst, ein Bereich für CSS Styles und die Logik der Komponente, welche aus JavaScript oder TypeScript Code besteht, befinden. Dies ermöglicht eine inhaltliche Zusammenfassung aller Teilaspekte eines Elementes der Webseite in einer einzigen Datei.

```
1      <template>
2          <span class="beispiel">Beispiel Komponente</span>
3      </template>
4
5      <script lang="ts">
6          import { defineComponent } from 'vue';
7
8          export default defineComponent({
9              name: 'Beispiel',
10         });
11     </script>
12     <style>
13         .beispiel {
14             background-color: lightgreen;
15         }
16     </style>
```

Listing 1: Beispiel einer simplen *vue component*-Datei

---

<sup>30</sup><https://vuejs.org/>

Durch diese Zusammenfassung wird der gesamte Code lesbarer, da meist nur eine Datei je gerade behandeltem Element des Frontends konsultiert werden muss. So wird Entwicklern viel Suchzeit und Arbeitsaufwand gespart werden, die anderweitig angefallen wären. Dies führte zu effizienterer und zufriedenstellenderer Arbeit am Projekt. Bessere Lesbarkeit führt auch zu langfristigerer Wartbarkeit. Dies ist in diesem Projekt besonders wichtig, da es unter Open-Source-Lizenz steht und gegebenenfalls von der Allgemeinheit weiter entwickelt werden soll. Mit adäquater Wartbarkeit ist eine Grundlage für langjähriges Überleben der *leek-box* geschaffen.

Ausmaß, Funktionalität und Größe von Komponenten sind allgemein keine Einschränkungen gesetzt. So kann es sich bei diesen zum Beispiel um Buttons, Header oder ganze Ansichten handeln. Für die Nutzung müssen diese allerdings noch mit *defineComponent* beim Framework registriert werden (siehe Listing 1). Danach können die Komponenten wie native HTML-Tags eingebunden werden. Dies macht die Nutzung von *components* sehr intuitiv. So hat das Framework eine sehr flache Lernkurve. Die Verwendung von weitreichend klassischem HTML, CSS und JS bzw. TS unterstützt dies zusätzlich. Für das Team war dies immens wertvoll und ein wichtiges Auswahlkriterium für die Wahl der Technologien, da für das Projekt nur wenige Monate Zeit zur Verfügung standen.

*Components* können auch innerhalb von anderen *components* genutzt werden. So lassen sich komplexere Elemente des Frontends aus kleineren, übersichtlicheren Komponenten zusammensetzen. Um dies zu ermöglichen, können von der aufrufenden Instanz Daten an die Komponenten übergeben werden. Anhand dieser können Inhalte, Aussehen und Verhalten dem Kontext entsprechend angepasst werden. Die Datenübergabe wird über sogenannte *properties* (kurz *props*) realisiert. Auch der Informationsfluss in die andere Richtung ist möglich und geschieht über *events*, welche von eingebundenen Komponenten erzeugt und ausgelöst werden können. Dabei ist es möglich Daten als Argumente an die *events* anzuhängen, damit die einbindende Komponente diese dann auslesen und weiterverwenden kann. Dies wird folgend in Listing 2 und 3 dargestellt.



```

1      <template>
2          <eintrag v-bind:name="'eins'" @eintrag-geklickt="
3              geklickt($event)" />
4          <eintrag v-bind:name="'zwei'" @eintrag-geklickt="
5              geklickt($event)" />
6          <eintrag v-bind:name="'drei'" @eintrag-geklickt="
7              geklickt($event)" />
8      </template>
9
10     <script lang="ts">
11         import { defineComponent } from 'vue';
12
13         import Eintrag from './Eintrag.vue';
14
15         export default defineComponent({
16             name: 'Liste',
17             components: {
18                 Eintrag,
19             },
20             setup() {
21                 function geklickt(wert: string): void {
22                     console.log(wert);
23                 }
24                 return { geklickt };
25             },
26         });
27     </script>

```

Listing 2: beispielhafte Listen-Komponente (Liste.vue)

```

1      <template>
2          <button @click="$emit('eintrag-geklickt', rückgabe)">{{
3              name }}</button>
4      </template>
5
6      <script lang="ts">
7          import { defineComponent } from 'vue';
8
9          export default defineComponent({
10             name: 'Eintrag',
11             props: {
12                 name: {
13                     type: String,
14                     default: 'Name',
15                 },
16             },
17             emits: ['eintrag-geklickt'],
18             setup(props) {
19                 const rückgabe: string = 'Eintrag ' + props.name + '
20                     ausgewählt';
21                 return { rückgabe };
22             },
23         });
24     </script>

```

Listing 3: beispielhafte Eintrag-Komponente (Eintrag.vue)

Hierbei gibt die Listen-Komponente die entsprechenden Namen an die *Eintrag*-Komponenten mit dem prop „name“ per *v-bind*: an die Einträge weiter. Die verwendete *property* „name“ wird in dem Eintragsselement angezeigt und beim Klicken auf dieses mit einem Event per *\$emit* an die Eltern-Komponente *Liste* zurückgesendet. Der mitgelieferte Wert des Events wird mit *\$event* ausgelesen und zur Veranschaulichung in der Konsole ausgegeben.

Alles in allem ermöglichte die Verwendung von *components* dem Team die Produktion von schlankem und übersichtlichen Frontend-Code.

## Reactivity

Reaktive Programmierung ist ein Programmierparadigma, welches Abstraktionen bereitstellt, um Werte, die sich nachträglich ändern können, zu nutzen und Abhängigkeiten zwischen diesen Werten abzubilden. [Bai+13] Als Paradebeispiel hierfür werden meist Tabellenkalkulationswerkzeuge wie zum Beispiel *Excel* angeführt. In diesen lassen sich Rechenergebnisse von den momentanen Werten anderer genutzter Zellen abhängig machen. Entsprechend ändert sich auch das Ergebnis automatisch, wenn Eingabewerte geändert werden.

Auch *Vue* ermöglicht es, diese Art von Funktionalität einzusetzen. So können

reaktive generische Datentypen z. B. in components verwendet werden. In diesem Projekt wurden dafür vor allem *reactive* für Objekte und *ref*<sup>31</sup> für primitive Datentypen verwendet. Wie das Beispiel in Listing 4 demonstriert, können so Datenänderungen sehr einfach angezeigt werden, da eine Referenz an Stelle einer eigentlichen Instanz übergeben wird, welche sich somit automatisch aktualisiert.

```
1      <template>
2      <div>
3          <span>{{ count }}</span>
4          <button @click="count ++">Increment count</button>
5      </div>
6  </template>
7
8  <script lang="ts">
9      import { ref } from 'vue';
10     export default defineComponent({
11         setup() {
12             const count = ref(0);
13             return { count };
14         },
15     });
16 </script>
```

Listing 4: Demonstration reaktiver Vue-Referenzen

In diesem Beispiel wird bei Instanziierung der Komponente (in *setup*) ein Zähler als reaktive Vue-Referenz (*ref*) angelegt. Dieser wird bei jedem Klick auf den Button inkrementiert. Da dies im HTML-Code passiert, wird der Wert hinter der Referenz direkt manipuliert. Dagegen muss im TypeScript-Code der Wert vorher noch mit *count.value++* „ausgepackt“ werden. Der geänderte Wert wird direkt auf der Webseite angezeigt, ohne dass diese neu geladen oder das DOM durch den Entwickler manipuliert werden muss. Stattdessen werden jeweils nur die grundlegenden Daten einer Komponente verändert und Vue bildet diese Daten mit Hilfe des Templates im DOM ab.

Dieses Paradigma findet auch in der Kommunikation zwischen verschachtelten *components* Anwendung. So sind alle *properties* von Komponenten reaktiv. Auch die an Events angehängten Daten können reaktiv sein. Dies ermöglichte dem Team in kurzer Zeit interaktive Benutzeroberflächen zu erstellen.

Ein weiterer, weniger zentraler Faktor für die Wahl des Frameworks *vue* war die Performanz. Dies zeigt sich im Vergleich zu anderen Web-Frameworks in Benchmark-Tests.[Car20] Für das Projekt war dies vor allem wichtig, da das System auf einem Raspberry Pi betrieben werden sollte und dementsprechend

<sup>31</sup><https://v3.vuejs.org/guide/reactivity-fundamentals.html>

nur begrenzte Leistung zur Verfügung steht.

### Tailwind CSS

Zur effizienteren Gestaltung der Benutzeroberfläche entschied sich das Team für das Utility-First-Framework *TailwindCSS*<sup>32</sup>. Dieses bietet im Vergleich zu anderen CSS-Frameworks wie *Bootstrap* mehr Flexibilität, da statt vorgefertigten Komponenten vielseitig verwendbare Utility-Klassen zur Verfügung gestellt werden, über die das Design definiert werden kann. So kann das Aussehen von Elementen innerhalb des HTML-Codes festgelegt werden. Damit steigt die Übersichtlichkeit des Frontend-Codes, da keine separaten CSS-Dateien und -Klassen angelegt werden müssen. [Sto20]

### FeathersJS

Bei der Erstellung des Backends entschied sich das Team zusätzlich das Framework *FeathersJS*<sup>33</sup> einzusetzen. Dies ermöglicht komfortable CRUD-Zugriffe<sup>34</sup> auf verschiedene Services, welche zum Beispiel Datenbanken oder externe APIs sein können. Durch eine Vielzahl von Adaptern können so zum Beispiel Daten in einer Datenbank ohne eigene Implementierung der Schnittstelle verwaltet werden. Es muss lediglich ein Service erstellt werden, der das generische Interface *Service* mit Typeparameter der zu verwaltenden Klasse implementiert. (vgl. Abb. 6) Dabei können verschiedenste Protokolle, wie HTTP oder WebSockets

```
class MyService implements Service<any> {  
  async find(params: Params) {}  
  async get(id: Id, params: Params) {}  
  async create(data: any, params: Params) {}  
  async update(id: NullableId, data: any, params: Params) {}  
  async patch(id: NullableId, data: any, params: Params) {}  
  async remove(id: NullableId, params: Params) {}  
}
```

Source: <https://docs.feathersjs.com/guides/basics/services.html>

Abbildung 6: Implementierung eines ServicesTest

zur Datenübertragung verwendet werden. Das WebSocket-Protokoll bietet hier den großen Vorteil, dass eine persistente Verbindung zwischen Server und Client (Benutzeroberfläche und Backend) besteht. So können sowohl Client, wie

---

<sup>32</sup><https://tailwindcss.com>

<sup>33</sup><https://feathersjs.com/>

<sup>34</sup>CRUD = Create-Read-Update-Delete

auch Server jederzeit mit der Datenübertragung beginnen, ohne vorher - wie bei HTTP - jedes Mal eine neue Verbindung aufzubauen zu müssen. [UK10] Dadurch kann der Server alle verbundenen Clients bei einem CRUD-Zugriff informieren, sodass alle ohne erneutes Anfragen der Daten den aktuellsten Zustand übermittelt bekommen. Wird also beispielsweise ein NFC-Tag von Person A geändert, sieht Person B diese Änderung ohne manuelle Aktualisierung der Seite. Eine weiteres Feature, welches vom Projektteam genutzt wurde ist die integrierte OAuth-Provider-Abstraktionen, die ein einfaches Authentifizieren mit Diensten, wie beispielsweise *Spotify* bei diesem Projekt ermöglicht. Dies war z.B. nötig, um die den NFC-Tags zugeordnete Musik abspielen zu können. Darüber hinaus können so auch Nutzerprofile angelegt werden, ohne persönliche Daten, wie z.B. Namen, Email-Adressen und Passwörter, speichern zu müssen. Nur Authentifizierungstokens des OAuth-Providers werden in der Datenbank hinterlegt. So konnte ein besserer Datenschutz für die Nutzenden ermöglicht werden.

### 5.2.3 Containervirtualisierung (Docker)

Es wurde lediglich die Anforderung gestellt, dass die Applikation vor allem für mobile Endgeräte optimiert sein soll. Auf Basis dieser Anforderung evaluierte das Projektteam bekannte Technologien und recherchierte mögliche Alternativen. Um die Installation der Box möglichst komfortabel zu gestalten, wird auf die freie Containervirtualisierungssoftware *Docker*<sup>35</sup> gesetzt. Ohne diese Möglichkeit wäre die Installation aufgrund der drei Microservices (Backend, Reverse-Tunnel und NFC-Reader) sehr aufwändig. Der Vorteil von Docker besteht darin, dass das Team lediglich eine sogenannte *docker-compose* Datei benötigt, in dem die Konfiguration der Docker-Container beschrieben ist. Es enthält einen Verweis auf die jeweiligen Container-Images, auf denen die Container aufbauen (z. B. leek-backend). Diese Images enthalten bereits alle notwendigen Abhängigkeiten und Programme, sodass weitere Installationen seitens der Benutzer:innen nicht erforderlich sind. Außerdem lässt sich das Verhalten der Images durch Umgebungsvariablen weiter konfigurieren. Durch die einfache Auslieferung bietet Docker außerdem den Vorteil der Reproduzierbarkeit. So können aufgetretene Fehler problemlos von einem Entwickler nachgestellt werden, da Docker das Betriebssystem des Anwenders von der benötigten Umgebung der leek-box abstrahiert.

---

<sup>35</sup><https://www.docker.com/>

#### 5.2.4 Reverse-Tunnel (ngrok)

Um verschlüsselt vom Frontend (unter <https://project-leek.github.io> erreichbar) auf das jeweilige Backend einer *leek-box* zuzugreifen, wird der Reverse-Tunnel Dienst *ngrok*<sup>36</sup> verwendet (siehe Abbildung 5). Dies ist notwendig, da moderne Browser eine verschlüsselte Verbindung (mit SSL-Zertifikat) zu allen Komponenten auf der Website verlangen, sobald die Website selbst per SSL geladen wird. Da das Framework jedoch nativ kein HTTPS unterstützt, wird *ngrok* verwendet, um die unverschlüsselten Daten durch einen verschlüsselten Tunnel zum Frontend zu schicken. Damit wird die Anforderung der Verschlüsselten Verbindung erfüllt. Außerdem soll das Backend unabhängig vom lokalen Netzwerk erreichbar sein, ohne dass eine umständliche Portweiterleitung eingerichtet werden muss. Um dies zu leisten, baut das Backend einen Tunnel zu dem Server mit der bekannten Adresse [ngrok.io](https://ngrok.io) auf. Dabei wird eine zufällige Subdomain im Schema *xyz.ngrok.io* angelegt und dem Benutzer in der Konsole des Backends angezeigt. Diese Adresse kann der Benutzer nun im Frontend (<https://project-leek.github.io>) eingeben, um sich mit seiner Box zu verbinden.

#### 5.2.5 Externe Bibliotheken

##### Spotify Web API

Um die abspielbaren Songs inklusive Cover-Bildern zu ermitteln und diese abzuspielen ist ein Zugriff auf die API eines Musikstreaming-Dienstes notwendig. Der vom Kunden vorgeschlagene Anbieter war *Spotify*<sup>37</sup>. Um den Zugriff auf die Spotify API zu simplifizieren wurde die Bibliothek *Spotify Web API Node*<sup>38</sup> verwendet. Diese bietet bereits die benötigten Methoden, wie z. B. die zum Suchen eines Songs anhand eines Suchbegriffs. Neben dem Songtitel und der Spotify-Url werden auch die Künstler und die Adresse des Albumcovers mitgeliefert und konnten von den Entwicklern ohne Mehraufwand genutzt werden. Auch eine Methode zur Ermittlung der verfügbaren Geräte zum Abspielen der Musik stellt die Bibliothek bereit.

##### NeDB

Als Datenbank für dieses Projekt wurde die kostenlos verfügbare JavaScript-Datenbank *NeDB*<sup>39</sup> gewählt. Sie wird verwendet, um die NFC-Tags, Benutzer und den angeschlossenen NFC-Reader zu verwalten.

---

<sup>36</sup><https://ngrok.com/>

<sup>37</sup><https://www.spotify.com>

<sup>38</sup><https://github.com/thelinmichael/spotify-web-api-node>

<sup>39</sup><https://github.com/louischatriot/nedb>

*NeDB* ist eine auf *MongoDB*<sup>40</sup> aufbauende, sehr schnelle JSON Datenbank. Sie wurde gewählt, da sie eine geringe Komplexität besitzt und weil *FeathersJS* einen Datenbankadapter für diese Datenbank bereitstellt, was den Zugriff auf die Datenbank sowie die Verwaltung der Daten erleichterte. Außerdem arbeitet die Datenbank inkrementell, was das Risiko auf sehr große Datenbankdateien minimiert. Diese Eigenschaft war hilfreich, um die Datenbank auf einem Raspberry Pi betreiben zu können. Zu Beginn des Projekts diskutierte das Team die Technologien, mit denen die *leek-box* umgesetzt werden sollte. Der Kunde ließ dem Projektteam hierbei sehr viele Freiheiten.

### 5.3 Hardware

#### Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, welcher ein Ein-Chip-System (SoC) mit einer ARM-CPU enthält. Durch das gute Preis-Leistungsverhältnis, die einfache Programmierbarkeit und der sehr guten Verfügbarkeit sind Raspberry Pis sehr beliebt und haben sich für dieses Projekt als Hardwarewahl durchgesetzt. Hierbei fungierte der Pi für das Hosting des Backends sowie der Anbindung an einen NFC-Reader.



Abbildung 7: Raspberry Pi

#### NFC Tags & NFC Reader

Ähnlich wie bei Konkurrenzprodukten, wird als Peripheriegerät für den Raspberry Pi ein NFC Reader genutzt. Die Steuerung der Box kann so mittels NFC-Tags durchgeführt werden. NFC steht für „Near Field Communication“ (Nahfeld-Kommunikation), ein Standard, der Datenübertragung auf sehr kurzer Distanz ermöglicht. NFC-Tags sind kleine Chips, die von einem NFC Reader ausgelesen werden können. Die Daten können je nach NFC Chip, von nur einer simplen *Id* bis hin zu beschreibbaren NFC Tags mit unterschiedlichsten Daten variieren. In diesem Projekt wurden die primitiveren



Abbildung 8: NFC Reader und NFC Tag

---

<sup>40</sup><https://www.mongodb.com/>

NFC-Tags, welche nur eine ID speichern, verwendet.

## 6 Durchführung

### 6.1 Meilenstein 1

Die Laufzeit des ersten Meilensteins erstreckte sich vom 29.10.2020 bis zum 11.11.2020 und war auf die Planung, Struktur und das Vorgehen des Projektes fokussiert. Zum Ende des Meilensteins sollte die Struktur und das Grundgerüst für das Projekt fertiggestellt sein. Hierfür sollte das Konzept der Software-Architektur sowie die genutzten Technologien (später Technologie-Stack genannt) auf Basis einer Anforderungsanalyse mit dem Kunden festgelegt werden. Außerdem sollten die ersten Tickets in das Backlog eingetragen werden. Auch musste das Projekt dahingehend geplant werden, dass jedem Teammitglied durchgehend zu bearbeitende Arbeitspakete zur Verfügung standen. Um einen schnellen und reibungslosen Start für die Entwicklung zu gewährleisten, sollte das Repository entsprechend vorbereitet werden. Das Team musste das Projekt so entsprechend planen, dass jedes Teammitglied durchgehend sinnvoll arbeiten konnte. Durch bereits vorhandene Erfahrung mit verschiedenen Technologie-Stacks und ähnlichen Problemen, wurde sich auf den Technologie-Stack geeinigt. Die verwendeten Technologien sind dem Abschnitt 5.2 zu entnehmen. Um für jedes Teammitglieds ein sinnvolles kontinuierliches Arbeiten zu gewährleisten, einigte sich das Team auf einen ständig gefüllten Ticket-Pool im Repository, sodass jederzeit selbstständig eine Aufgabe gefunden und bearbeitet werden konnte. Das Ergebnis des ersten Meilensteins bestand, abweichend von den späteren, nicht in einem Produkt Inkrement, sondern legte die Grundbausteine für die Entwicklung und das Teamwork. Neben dem bereits erwähnten Technologie-Stack, wurde als Host für die Applikation ein Raspberry Pi gewählt. Für die Entwicklung wurde ein Mono-Repository mit *lerna*<sup>41</sup> aufgesetzt, welches Front- und Backend zusammenfasste. Durch die Anforderung der Veröffentlichung als Open Source Projekt wurde eine verständliche und hilfreiche Dokumentation nötig, die einem Benutzer die Installation und Verwendung der *leek-box* aufzeigt. So wurde dem Repository eine einleitende Readme und ein „How to Contribute“ Guide hinzugefügt. Nach dem ersten Meilenstein fand noch keine nennenswerte Retrospektive statt.

---

<sup>41</sup><https://lerna.js.org/>



## 6.2 Meilenstein 2

Der zweite Meilenstein wurde für den Zeitraum vom 11.11.2020 bis zum 25.11.2020 angesetzt. Ziel war es, die ersten Grundbausteine für das Projekt legen. Damit das Team ein generelles Verständnis des Technologie-Stacks entwickeln konnte, sollte jedes Teammitglied eine „Hello-World“-Übungsaufgabe absolvieren. Diese beinhaltete das Anlegen eines Feathers-Services für ein virtuelles Haustier im Backend und die Darstellung im Frontend. Auch die Grundstruktur für den Bericht zu dem Projekt sollte in diesem Milestone erstellt werden, um den Fortschritt, Probleme und Erkenntnisse zeitnah dokumentieren zu können.

### Hello-World (Pet)

Zur Umsetzung setzten sich die Entwickler mit der Dokumentation von Feathers auseinander um einen Überblick über die notwendigen Klassen und Konstrukte zu erhalten. Zuerst musste eine Klasse erstellt werden, in der der Aufbau des Haustiers modelliert wurde. Sie beinhaltete Attribute wie „Name“ oder „Id“. Auf dieser Basis wurde anschließend ein Feathers-Service in Form einer Klasse erstellt. Diese implementierte das von *Feathers* bereitgestellte generische Interface *Service<T>*. Als Typparameter wurde hier die zuvor erstellte Model-Klasse angegeben. Um z. B. Objekte der Klasse *NicksPet* zu verwalten wurde folgender Service angelegt:

```
1      class NicksPetService implements MyService<NicksPet> {
2          id! : number;
3          name! : string;
4
5          constructor(_id: number, _name: string) {
6              this.id = _id;
7              this.name = _name;
8          }
9      }
```

Listing 5: NicksPetService (Hello World)

In dieser Klasse war außerdem ein Verweis auf eine weitere erstellte Klasse (Model) aufgeführt, welche bestimmt, mit welcher Schnittstelle oder API die Daten verwaltet werden sollen (in diesem Fall mit der durch *Feathers* bereitgestellten Schnittstelle für die Datenbank *NeDB*). Mit dem Erstellen dieser drei Klassen (*NicksPet.class.ts*, *NicksPetService.ts* und *NicksPet.model.ts*) waren die benötigten Komponenten zur Verwaltung vorhanden und die Arbeit im Backend abgeschlossen. Der nächste Schritt

beinhaltete die Implementierung der Steuerelemente zum Verwalten der Daten im Frontend. Aufgrund der erhöhten Komplexität der Einarbeitung in mehrere teilweise komplett unbekannte Frameworks zur selben Zeit, bestanden bei dieser Aufgabe einige Startschwierigkeiten, welche allerdings durch die große Hilfsbereitschaft der erfahreneren Teammitglieder überwunden werden konnte.

Im Frontend wurde pro Haustier je ein *component* zur Anzeige eines Haustiers angelegt, welches zusammen mit den Steuerelementen zum Anlegen, Löschen und Bearbeiten in einer *View* platziert wurde. Aufgrund der je nach Entwickler sehr unterschiedlichen Implementierung der *component* wird hier auf eine ausführliche Beschreibung verzichtet.

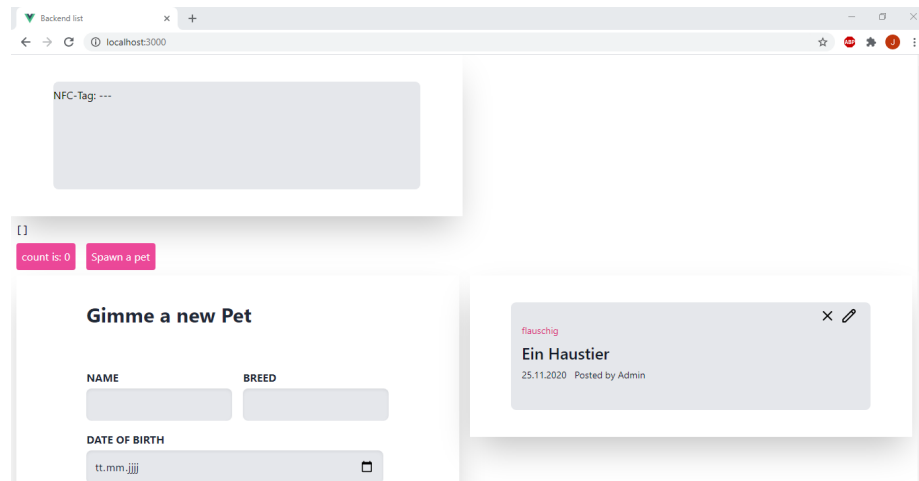


Abbildung 9: Zwei der Haustiere und die Anzeige des NFC-Readers

### NFC-Tags auslesen

Neben der Umsetzung des Hello-Worlds sollte sich mit dem Raspberry Pi und dem NFC-Reader auseinandergesetzt und das Auslesen der NFC-Tags implementiert werden. Diese Anforderung geriet mangels ausreichender Notizen beim ersten Review vorerst in Vergessenheit, sodass sich das Team einige Tage vor Sprintende nicht sicher war, ob diese Anforderung bis zum Review umgesetzt werden könnte. Um das Sprintziel nicht zu gefährden, mussten gegen Ende des Sprints Überstunden geleistet und das Feature konnte ins Produktinkrement aufgenommen werden.

Zur Umsetzung wurde im Backend die Klassen und der Service zur Verwaltung eines NFC-Readers und NFC-Tags analog zu dem in 6.2 beschriebenen Vorgehen erstellt. Außerdem wurde der Microservice *nfc-reader* in der

Projektstruktur unter *packages* angelegt. Dieser nutzt die Bibliothek *Evdev*<sup>42</sup>, welches ein Interface für die Events der angeschlossenen Eingabegeräte vom Linux-Kernel, wie in unserem Fall den NFC-Reader, bereitstellt. Dies ist möglich, da der NFC-Reader sich wie eine Tastatur verhält. Er simuliert dabei eine Tastatureingabe mit dem Text der NFC-Tag-ID. Zur Nutzung wurden zwei Klassen erstellt. Zum einen die NFC-Reader Klasse (im NFC-Reader Package), welche jeden simulierten Tastendruck des NFC-Readers abfängt, bis ein Zeilenumbruch (0A HEX) empfangen wird und anschließend ein Event auslöst, welches die eingescannte ID beinhaltet. Die Hauptklasse nimmt dieses Event entgegen (Event-Listener) und setzt mittels des NFC-Reader Services den aktuell anliegenden Tags (*currentTag*) auf die übergebene ID. Durch die Nutzung von *Web-Sockets* und *Vue.js* kann diese Änderung sofort im Frontend angezeigt werden. Aufgrund der potentiell hohen Komplexität des Abspielens von Musik wurde diese Funktionalität für den folgenden Milestone eingeplant und vorerst lediglich die Anzeige der NFC-ID als JSON-String umgesetzt. Für den Microservice wurde anschließend ein Docker-Image kreiert, welches eine einfache und gleichzeitig optionale Auslieferung zur restlichen Installation ermöglicht. Dies bringt den Vorteil, dass auch auf einem anderen System (wie z. B. einem esp32) integrierte NFC-Reader, statt des aktuellen genutzt werden können.

### Prototyping der Beutzeroberfläche

Damit die Entwicklung zeitnah starten konnte, wurden die ersten Mockups für die Benutzeroberfläche erstellt. Mithilfe des Prototyping-Tools *figma* (vgl. 4.1.3) wurden Prototypen für die Start-, die Hauptansicht und die Ansichten für das Bearbeiten und Anlegen von NFC-Tags erstellt (Eine Übersicht über das erstellte Design ist im Anhang unter 8 zu finden). Nach dem Design wurden die Ansichten in *Click-Dummys* umgewandelt, um dem Kunden im folgenden Review einen Überblick über das Verhalten der Benutzeroberfläche zu verschaffen.

### Review

Dank des großen Engagements und der Bereitschaft Überstunden zu leisten, konnte dem Kunden zum Review ein neues Produktinkrement übergeben werden. Das Produkt wurde um ein erstes User Interface erweitert, welches die Tag-ID vom angelegten NFC-Tags anzeigen konnte. Der NFC-Reader wurde mit dem Raspberri Pi verbunden und als ein auslieferbares Docker Image bereitgestellt. Auch wurden für die Entwicklung automatisierte Tests erstellt,

---

<sup>42</sup><https://github.com/PixnBits/node-evdev>

welche neuen Code auf etablierte Code-Konventionen und Lauffähigkeit überprüften.

### **Retrospektive**

Vor allem der Einsatz und das Know-How vom Teammitglied Anton wurde hier wertgeschätzt, welcher sich um die Automatisierung von Tests und Deployment kümmerte. Auch das Gruppenklima wurde sehr gelobt und angemerkt, dass die Zusammenarbeit, Kommunikation und die regelmäßigen Meetings sehr positiv wahrgenommen wurden. Das Team konnte bei der Umsetzung dieses Meilensteins viel über die Funktionsweise der Frameworks *Feathers*, *Vue.js* und *Tailwind* lernen. Darüber hinaus kamen viele das erste mal Automatisierungsprozessen, Code-Reviews und Pull Request in Kontakt. Das Team bemängelte bei dem Meilenstein aber auch, dass der Kundenwunsch zu spät behandelt wurde und die Prioritäten falsch gesetzt wurden. Somit wurde als Verbesserungen für die nächsten Meilensteine eine bessere Koordination und Aufgabenverteilung vorgeschlagen.

## **6.3 Meilenstein 3**

Vom 25.11.2020 bis zum 17.12.2020 fand die Umsetzung von Meilenstein drei statt.

### **Mockup Überarbeitung**

Die erstellten Mockups für die Benutzeroberfläche sollten in diesem Sprint basierend auf dem Feedback von macio finalisiert werden. Das Ergebnis führte bei den Entwicklern anfänglich zu etwas Unsicherheit, da das Team nur aus Studierenden des Studiengangs Informationstechnologie bestand, welche mit dem Thema Usability, abgesehen von dem Modul *Usability Engineering*<sup>43</sup>, wenig Kontakt haben und darüber hinaus wenig persönliche Erfahrungen in dem Themenbereich gesammelt hatten. Diese Schwierigkeiten wurden durch offene Kommunikation an macio weitergegeben, die daraufhin einen ihrer Designer zur Unterstützung bei der Entwicklung bereitstellten.

---

<sup>43</sup><https://moduldatenbank.fh-kiel.de/de-DE/Module/Details/0de99d45-efc9-437a-a190-24d539b2a1d8>

## Flow-Charts

Außerdem wurden die im letzten Sprint erstellten *User Stories* als Grundlage für die Entwicklung von *Flow-Charts* verwendet. Diese wurden mit dem Online-Tool *Lucidchart* (vgl. 4.1.4) erstellt und sind im Anhang unter 8 zu finden. Sie erleichterten die Umsetzung der Design-Mockups, indem sie notwendigen Abläufe visualisierten. Dank dieses Zwischenschrittes konnten auch die nicht durch Mockups visualisierte Funktionen, wie das Verbinden der Box mit dem WLAN, zuerst logisch skizziert werden, bevor sie programmiert wurden. Dadurch konnte innerhalb des Teams ausführlich die beste Methode diskutiert werden, um die bestmögliche Usability für den Benutzer sicherzustellen. Außerdem konnten sie auch dem Kunden vorgelegt werden, um etwaige Misskommunikation bei der Anforderungsanalyse zu vermeiden und so die Wahrscheinlichkeit auf spätere Änderungen wichtiger Funktion zu senken. Die Erstellung dieser *Flow-Charts* hat mit übereinstimmender Meinung aller Teammitglieder die spätere Programmierung maßgeblich effizienter gestaltet und wurde auch vom Kunden sehr geschätzt.

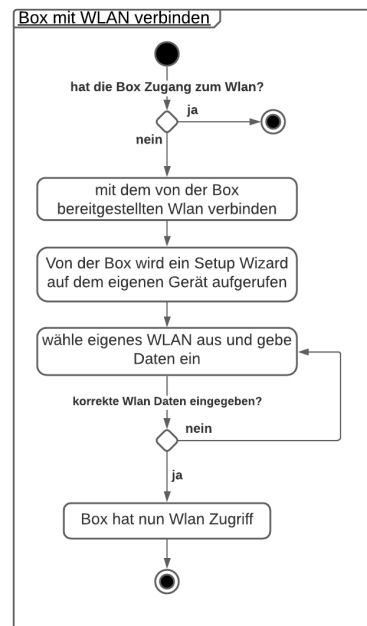


Abbildung 10: Flowchart: Connect to WIFI

## Musik abspielen

Auch der vom Kunden geäußerte Wunsch, beim Einlesen eines NFC-Tags die entsprechende hinterlegte Musik abzuspielen, sollte in diesem Milestone umgesetzt werden. Dafür setzte sich ein Teil des Teams intensiv mit der *Spotify-API*<sup>44</sup> auseinander. Hierbei entdeckten die Entwickler die Bibliothek *Spotify Web API node* (vgl. 5.2.5), welche den Zugriff auf die *Spotify-API*, abstrahiert und so den Zugriff auf diese erleichtert. Die Bibliothek wurde in Kombination mit *Feathers-Hooks*<sup>45</sup> verwendet. Hooks sind Funktionen, die z. B. vor oder nach einem Feathers Service Aufruf ausgeführt werden können. In diesem Fall beinhaltet der angelegte *playSpotify* Hook einen Aufruf der Methode *play* der *Spotify Web API Node*, die den Song hinter der *URL* des

<sup>44</sup><https://developer.spotify.com/documentation/web-api/>

<sup>45</sup><https://docs.feathersjs.com/api/hooks.html>

mitgegebenen NFC-Tags abspielt. Dieser Hook ist an die *patch* Methode des NFC-Reader Services angehängt, sodass dieser im Anschluss an die *patch* Methode, die den Eintrag des *attachedTag* des NFC-Readers aktualisiert, aufgerufen wird.

### NFC-Tags scannen emulieren

Da während des Projekts nicht alle Entwickler kontinuierlichen Zugang zu einem NFC-Reader hatten wurde im Zuge dieses Milestones außerdem die Möglichkeit geschaffen, das Scannen eines NFC-Tags zu emulieren. Hierfür wurde ein *Shell-Script* erstellt, welches einen *HTTP PATCH request* an das Backend schickt. Dadurch wird der *attachedTag* des NFC-Readers auf Basis eines übergebenen Parameters gesetzt. So kann zum Beispiel ein NFC-Tag mit der ID 12345 emuliert werden, indem im Terminal der Aufruf `./simulate-read.sh 12345` durchgeführt wird, ohne dass der Entwickler einen physischen NFC-Reader an seiner Entwicklungsmaschine angeschlossen haben muss. So konnten Funktionen wie das Anlegen eines neuen NFC-Tags von jedem Entwickler durchgeführt werden.

### Entwicklung des Welcome Screens mit Authentifizierung

Um das in diesem Milestone erstellte Feature des Abspielens von Musik sinnvoll nutzen zu können, musste sich auch mit der Authentifizierung mit Spotify beschäftigt werden. Dafür wurde das offene Protokoll *OAuth* verwendet, welches bereits in *Feathers* abstrahiert implementiert ist. Zur Nutzung dieses Protokolls wurde eine sogenannte *Strategy* angelegt, welche den für die Anmeldung notwendigen Datenaustausch mit Spotify ermöglicht. Die dafür nötige *Strategy* mit dem Namen *SpotifyStrategy* wird durch eine Klasse realisiert, welche von der Feathers bereitgestellten Klasse *OAuthStrategy* erbt. Bei der Anmeldung eines Benutzer wird von Spotify ein JWT-Token an das Backend übergeben. Ein JWT-Token besteht aus einem Base64 enkodiertem Text, welcher beliebige Daten beinhalten kann.

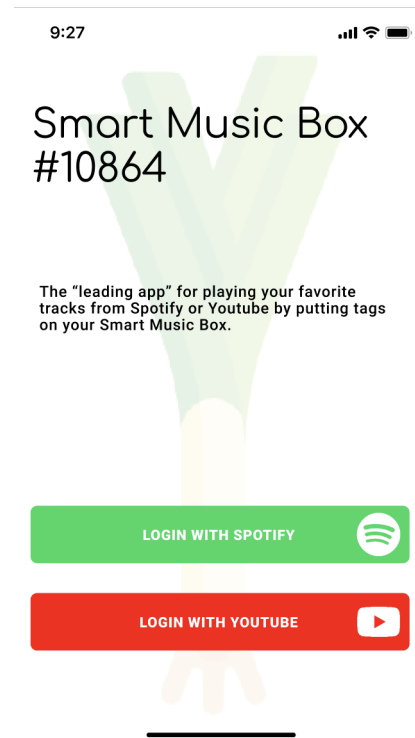


Abbildung 11: Welcome Screen  
Version 1

Diese Daten sind unverschlüsselt (nur enkodiert) in dem Token hinterlegt und werden durch eine Signatur auf ihre Integrität überprüft. Der von Spotify übermittelte JWT-Token enthält in diesem Fall die ID, E-Mail und den Namen des Benutzers. Diese Daten werden extrahiert und sofern der Benutzer nicht bereits in der Datenbank vorhanden ist, zum Anlegen eines neuen Benutzers verwendet. Sollte der Benutzer jedoch bereits in der Datenbank existieren, so wird dieser gegebenenfalls aktualisiert. Da sich jeder Benutzer vor der Verwendung der App zuerst mit einem gültigen Token identifizieren muss, wird bei möglichen Änderungen der Stammdaten in Spotify sofort immer der Datenbankeintrag in Feathers aktuell gehalten.

Zum Beginn der Umsetzung wurde auf Basis des Mockups ein View in der Benutzeroberfläche (*app* package) erstellt. Dieser enthält als Steuerelemente Infos über den Namen des Produkts - *leek-box* - und einen kurzen Untertitel, sowie einen „Anmelden mit Spotify“ Button, der den OAuth-Prozess startet. Diese Seite wurde mittels des *Vue Routers* so konfiguriert, dass sie nur aufgerufen wird, wenn der Benutzer nicht angemeldet, also kein JWT-Token vorhanden ist. Ist ein JWT-Token vorhanden, wird der Nutzer bei dem versuchten Aufruf dieser Seite automatisch auf die Hauptseite weitergeleitet.

### Veröffentlichung der App auf Github Pages

Zur Veröffentlichung unseres Frontends der *leek-box* verwenden wir den Dienst Github Pages welcher ein Webhosting von statischen Dateien unter einer *xyz.github.io* URL anbietet. Das Team verwendete für das Frontend ein separates Repository *project-leek/project-leek.github.io* mit der URL *project-leek.github.io*. Nach dem Continuous-Delivery Verfahren, baut die Pipeline des eigentlichen *project-leek* Repositories automatisch nach jedem Merge auf den master Branch das *app-package*. Dafür wird das Build-Tool des *app-packages* aufgerufen und die dabei entstandenen statischen Dateien werden von dem Pipeline Plugin *github-pages-deploy-action*<sup>46</sup> aus der Pipeline in das separate *project-leek/project-leek.github.io* Repository für den GitHub Pages Dienst kopiert. Somit befindet sich unter *project-leek.github.io* immer die aktuellste Version der App. Dies hat den Vorteil, dass Änderungen wie zum Beispiel Bugfixes und neue Features dem Endanwender schnellst möglich zur Verfügung gestellt werden können ohne das auf einen Release-Plan Rücksicht genommen werden muss.

### Review & Retrospektive

---

<sup>46</sup><https://github.com/JamesIves/github-pages-deploy-action>

Zusätzlich zu dem Wunsch, beim Anlegen eines NFC-Tags die jeweilig hinterlegte Musik abzuspielen, war zum Review auch das Einloggen in einen bestehenden Spotify Account möglich. Jegliche Neuerung auf dem Master wurden außerdem auch auf Github.io veröffentlicht, sodass der Kunde jederzeit den Fortschritt ermitteln konnte.

Die vom Team durchgeführte Retrospektive ergab, dass die Problembewältigung durch die Anwendung von Pair Programming in diesem Sprint optimiert werden konnte. Auch der Workflow mit Github und den Pull Request wurde verinnerlicht sowie Code Reviews verstärkt durchgeführt. Das Team lernte in diesem Sprint Vue Hooks und die Stärken von Feathers in Verbindung mit OAuth kennen. Festgestellt wurde außerdem, dass einige Tickets weiterhin stark voneinander abhängig waren, weshalb das Bearbeiten einiger Aufgaben nicht direkt möglich war. Da Abhängigkeiten teilweise unvermeidbar sind, wurde sich geeinigt, diese Tickets bei hoher Komplexität mittels Pair-Programming umzusetzen, um sie möglichst schnell und qualitativ umsetzen zu können. Um das Risiko der Entstehung zukünftiger Probleme zu senken wurde außerdem festgelegt mögliche Problemstellungen nicht erst im Standup, sondern so früh wie möglich mitzuteilen. Genauso sollte neu gelerntes besser kommuniziert werden, um die Arbeit effizienter zu gestalten. Hierfür wurden die *Standups* als Rahmen gewählt.

## 6.4 Meilenstein 4

Zu Beginn dieses Meilensteins waren alle Grundlagen gelegt um mit der Umsetzung der Hauptfunktionen zu beginnen. Es sollten möglichst viele Teile des designten User Interfaces umgesetzt werden, allerdings lag der Fokus auf einem vollständigem „Add-Tag-Flow“. Dazu gehörten:

1. Scannen des Tags
2. Vergeben eines eigenen Namens
3. Auswählen von Musik
4. Auswählen eines eigenen Bilds

Weiterhin sollten für den nächsten Schritt die Mockups erstellt werden, wie ein Nutzer den Lautsprecher zur Wiedergabe auswählt sowie seine eigene *Leak Box* initial einrichtet.

Um paralleles Arbeiten und Eigenständigkeit der Tickets zu ermöglichen, wurde jede Ansicht in ihre einzelnen Komponenten aufgeteilt. Diese Komponenten sollten für sich alleine stehend funktionieren und so anpassbar sein, dass z.B. nur ein Button Komponent entwickelt werden musste, dass auf



allen Ansichten verwendet werden konnte. Es wurde daher erst die grundlegenden Komponenten wie Buttons, Textfields und Dropdowns entwickelt um aus diesen Komponenten Views zu erstellen.

Die Views wurden ebenfalls so in Tickets aufgeteilt, dass möglichst parallel gearbeitet werden konnte. Beispielsweise wurde die Darstellung auf dem Homescreen und die Auswahl von Infos, Musik und Bild komplett unabhängig voneinander bearbeitet und mit Platzhalterdaten befüllt. Schließlich wurden alle Komponenten und Seiten zusammengefügt um einen vollständigen Flow zu ermöglichen. Es konnte nun erfolgreich per UI ein neuer Tag der Datenbank hinzugefügt und in dem HomeScreen angezeigt werden.

### Components

Für Elemente, die häufig in der Anwendung genutzt werden (Button, Textfield, DropDownList), entwickelte das Team *Vue Components*, die in dem Ordner *uiBlocks* für alle Entwickler leicht zugänglich abgelegt wurden. Diese *components* garantieren, dass die Elemente anpassbar sind und dennoch einem einheitlichen Design folgen. Besonders am Anfang halfen sie dem Entwickler, da sie die Komplexität der Implementierung von Design und Funktionalität senkten. Diese mussten lediglich einmalig in der *component* definiert werden und konnten dann bei Bedarf in den *Views* eingebunden werden. Es musste dann nur noch der Inhalt über *properties* übergeben werden. Während

der fortschreitenden Entwicklung trat allerdings das Problem auf, dass immer mehr Anforderungen an die *components* gestellt wurden, und damit immer mehr *Properties* für deren Anpassung notwendig waren. Dies resultierte darin, dass zum Beispiel die *Button-Component* sehr unübersichtlich wurde und redundanten Code enthielt. Aufgrund der verschiedenen geforderten Größen und Anforderungen der enthaltenden Texte und Bilder, stieg die Anzahl der zu übergebenen Parameter immens, was zu deutlich niedriger

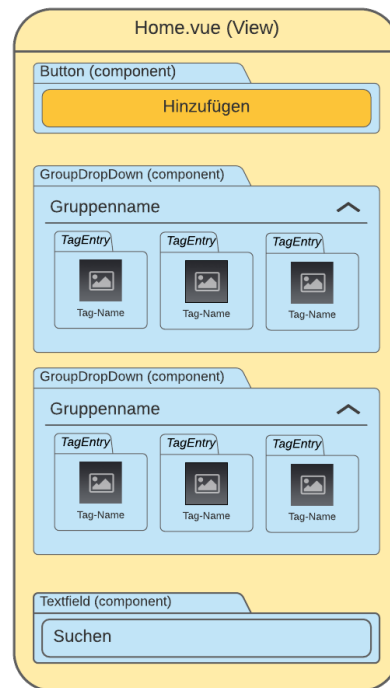


Abbildung 12: Schematischer Aufbau Home.vue aus *components*

Übersichtlichkeit führte. Als Lösung wurde in späteren Meilensteinen eine erneute Analyse durchgeführt, um zu ermitteln, welche Anforderungen einzelne Komponenten erfüllen mussten. Die Ergebnisse dieser Analyse wurden anschließend genutzt, um den Code zu überarbeiten, wodurch die Übersichtlichkeit der *components* wiederhergestellt wurde.

### Umsetzung des Homescreens

Im HomeScreen sollten alle angelegten Tags gruppiert angezeigt werden. Hierfür wurden zwei *components* angelegt. Die Erste (*TagEntry*) enthält das Bild und den Namen eines NFC-Tags und kann in der Anwendung vielseitig wiederverwendet werden. Die Zweite *component* mit dem Namen *GroupDropDown* repräsentiert eine Gruppe, zu der ein NFC-Tag gehört. Pro NFC-Tag in der jeweiligen Gruppe wird eine Instanz der *TagEntry component* in die Gruppe geladen. Zusätzlich erhielt die *GroupDropDown component* die Funktionalität auf- und zuklappbar zu sein. Diese *components* werden in *Home* angezeigt. Neben der Einbindung von einer *GroupDropDown component* pro Gruppe wurde im Kopfbereich noch die vom Team erstellte *Button component* verwendet, die das Anlegen eines neuen NFC-Tags einleiten sollte. Im Fußbereich befindet sich eine Suchleiste, die das Filtern der angezeigten NFC-Tags ermöglichen soll. Die beiden letztgenannten Features waren zu diesem Zeitpunkt noch nicht umgesetzt.

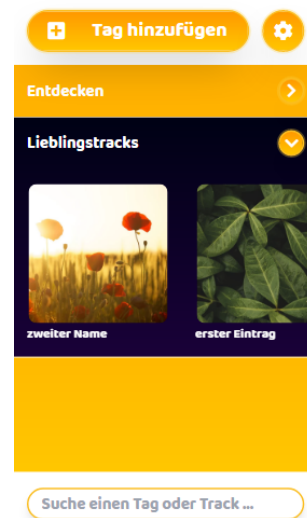


Abbildung 13: Home View

### Entwicklung der „Tag hinzufügen“ Funktionalität

Um dem Benutzer das Hinzufügen von neuen NFC-Tags zu ermöglichen, wurde eine neue *View* mit dem Namen *AddTag* erstellt, welche wie auch die *Home-View* in Kopf-, Haupt- und Fußbereich eingeteilt ist. Beim Laden der *View* wird eine neue Instanz der Klasse *NFC-Tag* erstellt.

```

1      <component
2          :is="steps[activeStep].component"
3          v-model:nfc-tag="nfcTag"
4          @update:is-valid="dataValid = $event"
5          @proceed="nextStep"
6      />

```

Listing 6: Dynamisches Laden der components von AddTag

Im Hauptbereich der *View* werden die vier *components*, die für die Eingabe der NFC-Tag Daten zuständig sind, dynamisch geladen. Hierfür wird der beim Laden erstellte NFC-Tag (*newTag*) an alle *components* nacheinander weitergegeben und in ihnen mit Informationen gefüllt. Der Fußbereich enthält abhängig von der im Hauptbereich gerade aktiven *component*, *Button components* für „Zurück“, „Weiter“ und „Anlegen“. Alle der folgenden *components* beinhalten Validierung für die eingegebenen Daten, sodass sichergestellt wird, dass das *newTag* jederzeit gültige Daten besitzt. Bei Änderung der Werte in den *components* wird das Event *update:is-valid* ausgelöst und in *AddTag* die *Disabled*-Eigenschaft des sich in der Fußzeile befindliche Button abhängig vom der Gültigkeit verändert. So kann in jedem Schritt nur fortgefahren werden, wenn alle Eingaben die korrekte Form haben. Alle diese *components* werden ebenfalls für die Bearbeitung der NFC-Tags wiederverwendet.

Die Erste *component*, welche in den Hauptbereich geladen wird, ist *TagStepPlaceTagOnReader*. Dieser beinhaltet einen Text und ein *GIF*, welches den Benutzer zum Scannen des NFC-Tags auffordert. Im *TypeScript*-Code wird im *Lifecycle Hook onMounted* an das *patch*-Event des NFC-Readers der Aufruf der Funktion *attachedTagListener* gebunden. Wird nun ein NFC-Tag gescannt, führt der *NFC-Reader Microservice* mittels Feathers ein Update (Aufruf des *Patch*-Events) auf den *NFC-Reader* durch und setzt das Attribut *attachedTag* auf die gescannte NFC-ID. Anschließend wird geprüft, ob ein NFC-Tag mit dieser ID bereits in der Datenbank vorhanden ist und der Benutzer in diesem Fall an die *View* zum Bearbeiten des Tags weitergeleitet. Ist der NFC-Tag noch nicht angelegt, wird die ID an die *ID-Property* des *newTag* gebunden und die



Abbildung 14: Step Scan (Add)

Events *update:nfc-tag* und *proceed* ausgelöst. *update:nfc-tag* bewirkt, dass das Attribut *TagId* des *newTag* gesetzt wird. *proceed* weist *AddTag* an, die nächste *component* in den Hauptbereich der *View* zu laden. Dabei wird die Funktion *attachedTagListener* vom *patch*-Event abgemeldet.

Als Nächstes wird die *TagStepInfo* *component* geladen. Sie beinhaltet eine *Textbox* *component* für den Namen des NFC-Readers. Diese erhält als *v-model* eine *computed property*<sup>47</sup>. Diese ruft im *setter* die Methode *updateTag* auf, welche die zwei Parameter *key* und *value* erwartet, um die Eigenschaft *key* (in diesem Fall „name“) auf den im Parameter *value* übergebenen Wert zu setzen. Als Zweite *component* kommt ein *DropDown* zum Einsatz, welches die Auswahl der Gruppe ermöglicht. Hierbei kann sowohl aus bereits bestehenden Gruppen gewählt, als auch direkt in der *component* eine neue erstellt werden. Auch hier wird die Methode *updateTag* verwendet, die in diesem Fall mit dem Wert „group“ für den Parameter *key* aufgerufen wird. Der eingegebene Name und die Gruppe werden an die jeweiligen Eigenschaften von *newTag* gebunden und das Event *update:nfc-tag* aufgerufen, wodurch die *View AddTag* den Wechsel zur nächsten *component* einleitet.

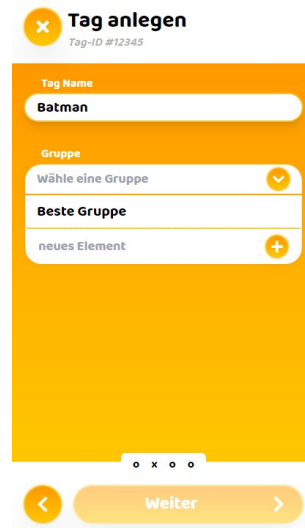


Abbildung 15: Add Step Scan-Tag

<sup>47</sup><https://v3.vuejs.org/guide/reactivity-computed-watchers.html>

Hier in der *TagStepTrack* erhält der Benutzer die Möglichkeit, den Song auszuwählen, der beim Scannen des NFC-Tags abgespielt werden soll. Der gewünschte Song-Titel oder Suchwörter können in eine *Textbox component* eingegeben werden. Zum Abrufen der Songs von Spotify wurde der *SpotifyTrackService* erstellt, in dem die Methode *find* überschrieben wurde. Diese beinhaltet den Aufruf der Funktion *searchTracks* der *Spotify Web Api*, welcher als Parameter der in der Suchtextbox eingegebene Text übergeben wird. Für jeden zu dem übergeben Text passenden Titel wird dann eine Instanz der erstellten Klasse *Track* erstellt, die Informationen über die *URL*, den Titel, die *Bild-URL*, und die Künstler erhält. Das Array mit den passenden *Track*-Instanzen wird anschließend an *TagStepTrack* zurückgegeben und die Infos mit Bild, Titel und Interpreten angezeigt. Die Auswahl erfolgt dann per Klick auf den gewünschten Song. Bestätigt wird die Auswahl per Klick auf „Weiter“ in der Fußzeile. Dadurch wird *update:nfc-tag* Event ausgelöst und die letzte *component* in die *AddTagView* geladen.

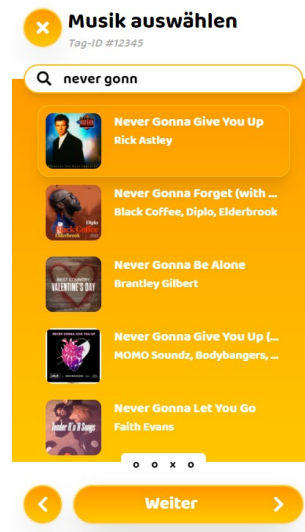


Abbildung 16: Add Step Track

Zuletzt kann im *TagStepImage* das gewünschte Bild für den NFC-Tag ausgewählt werden. Standardmäßig wird hier das Albumcover des Songs ausgewählt, welches in *AddStepTrack* bereits gesetzt wurde. Um sicherzustellen, dass die Albumcover jederzeit aktuell sind, wird in der Datenbank nicht das Bild, sondern lediglich die URL des Bildes gespeichert. Sollte der Musik-Streaming-Anbieter dieses ändern, wird so automatisch auch das Albumcover in der Applikation geändert. Dies hat außerdem den Vorteil, dass die Anwendung wesentlich speicherplatzeffizienter ist, da statt teilweise sehr großen Bilddateien jeweils nur die URL gespeichert wird. Um dem Benutzer mehr Freiheiten in der Gestaltung zu geben, kann alternativ auch ein eigenes Bild verwendet werden. Dieses muss in der aktuellen Version auf einem

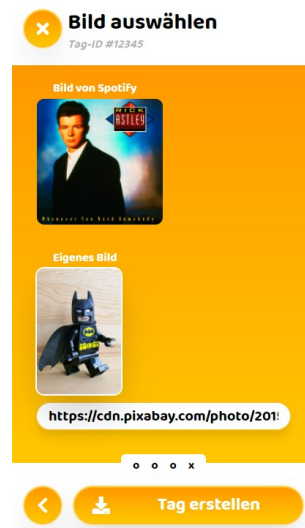


Abbildung 17: Add Step Image

Online-Dienst vorliegen. Ist dies gewünscht, kann in einer entsprechenden Textbox die *URL* dieses Bildes eingetragen werden, worauf das Bild automatisch zur Vorschau in ein dafür vorgesehenes Vorschaufeld geladen wird. Dies stellt sicher, dass der Benutzer nicht versehentlich eine falsche Bild-URL wählt. Zusätzlich wird diese Textbox zusätzlich mittels eines *RegExp* geprüft, um sicherzustellen, dass es sich um eine nutzbare Bilddatei handelt. Zur Bestätigung muss entweder auf das Albumcover oder auf das eigene Bild geklickt werden. Das Erstellen des Tags kann dann mit einem Klick auf „Anlegen“ im Fußbereich der *View* abgeschlossen werden. Durch die angestrebte Unabhängigkeit musste besonders viel Arbeit investiert werden, um die Features miteinander zu verbinden. Dieser Prozess dauerte länger als erwartet und forderte ein hohes Maß an Kommunikation zwischen den Teammitgliedern. Teilweise entstanden Wartezeiten und funktionierender Code wurde, über mehrere Merges, instabil oder defekt. Kurz vor dem Review musste noch viel getan werden, um eine reibungslose Präsentation zu garantieren.

Nachdem die ersten großen Merges abgeschlossen wurde, entschied das Team eine interne und anonyme Evaluation der Teammitglieder zu starten. Jedes Teammitglied hatte somit konstruktive Kritik aber auch Lob und Unterstützung erhalten und konnte an den eigenen Schwächen arbeiten und Stärken weiter ausbauen. Darüber hinaus wurde auch beschlossen, in den folgenden Sprints die Menge an zu bearbeitenden Issues besser festzulegen. Das Produkt wurde um die Grundfunktionen erweitert. Es ist nun möglich, Tags anzulegen und entsprechend den Namen, Musik und ein Bild festzulegen. Genauso ist es dem Nutzer nun möglich, alle angelegten Tags zu durchsuchen. Es wurde weiterhin auch das Design Mockup von dem von macio gestellten Designer in Vue größtenteils umgesetzt. Für die Entwickler wurden auch die automatisierten Tests um einen Test auf Typsicherheit erweitert. Besonders gut gefiel dem Team das Teamklima, vor allem das ehrliche Miteinander. Trotz Komplikationen und Probleme konnte dem Kunden am Ende ein funktionierendes Product Increment geliefert werden. Pull Requests und Tickets wurden besser beschrieben, sodass Probleme verständlicher waren. Vor allem durch das interne Team Review wurde die Zusammenarbeit weiter verbessert. Das Team konnte das Wissen zu Vue, vor allem die Themen Reaktivität, Events und Kommunikation zwischen Komponenten ausbauen. Insgesamt war das Zeitmanagement verbesserungswürdig, da sich für einzelne Tickets zu viel Zeit genommen wurde. Das Team stimmte überein, dass eine bessere Planung sowie Abstimmung für die nächsten Meilensteine notwendig wäre. Auch sollte eine frühere interne Deadline gesetzt werden, damit nicht kurz vor dem Sprintende noch viele Fehler auftreten und behoben werden

müssen. Um insgesamt auch die Arbeit kontinuierlicher zu gestalten, sollten Aufgaben in kleinere Probleme aufgeteilt werden.

## 6.5 Meilenstein 5

Mit dem fünften Meilenstein, der zwischen dem 14.01.2021 und 28.01.2021 stattfand, sollte die Entwicklung am Produkt weitgehend abgeschlossen werden, sodass der Fokus auf den Bericht gelegt werden konnte. Geplant war die letzten Anwendungsszenarien final umzusetzen, sodass die folgenden beiden Meilensteine zum Beheben kleinerer Fehler genutzt werden konnten. Somit war das Ziel die Umsetzung einer *View* für die Einstellungen der *Leek Box*, als auch der Tags. Außerdem sollten die User Interface Elemente, sofern noch nicht geschehen, ein einheitliches Design erhalten. Die Dokumentation zur Bedienung der *Leek Box* sollte fertiggestellt und mit der Arbeit am Bericht begonnen werden.

Die *View* für die Einstellungen der *Leek Box* wurde hinzugefügt, die durch ein Zahnrad auf der Startseite aufgerufen werden kann. In diesen Einstellungen können die Nutzer nun den Lautsprecher für die Wiedergabe auswählen und sehen, mit welcher Email-Adresse sie angemeldet sind. Außerdem enthält die Ansicht einen Button um sich auszuloggen.

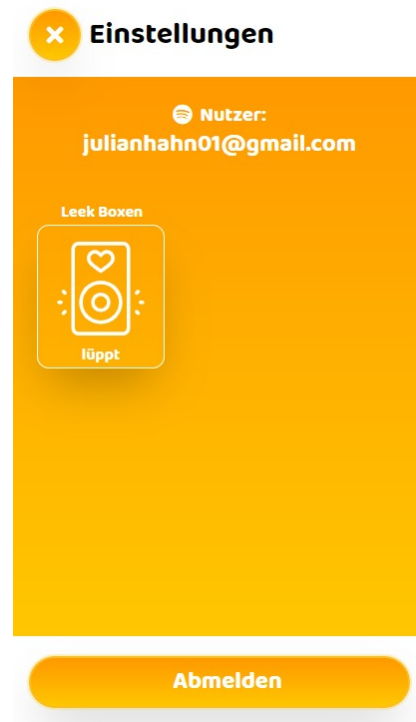


Abbildung 18: Leek Box Einstellungen

Auch die Funktionalität zum Bearbeiten der NFC-Tags wurde in diesem Meilenstein umgesetzt. An dieser Stelle zeigten sich erneut die Vorzüge von Frameworks wie *Vue*, da somit die bereits vorhandenen *components* wiederverwendet werden konnten. Um ein gleiches Design der Überschriften der einzelnen Inputs zu garantieren, wurden die erstellten *components LabeledInput* für die Anzeige von Namen, Gruppe, Musik und Bild verwendet. Zur Änderung des Namens wird das erstellte *component TextInput* genutzt, dem beim Aufruf der *View* eine Referenz auf *nfcTag.name* übergeben wird, sodass dieses den Namen anzeigt. Ähnlich erfolgt die Auswahl und Anzeige der Gruppe. Hier wird anstatt der *TextInput component* die *Dropdown component* verwendet, die eine Referenz auf *nfcTag.Group* empfängt und als Elemente die bereits existierenden Gruppen enthält. Der Benutzer hat ebenfalls die Möglichkeit neue Gruppen anzulegen. Für die Anzeige des dem Tag zugewiesenen Musikstücks wird ein dem *TagEntry component* visuell ähnliches Element verwendet, welches als Hintergrund das angedeutete Logo des Musikstreaming-Anbieters enthält. Zur Änderung des Musikstücks ist die Bearbeiten-Schaltfläche (*Button component*) vorgesehen, die den Benutzer auf die bereits beim Anlegen verwendete *component AddStepTrack* weiterleitet, die die benötigte Funktionalität liefert. Das gleiche Verfahren wird auch für die Anzeige und Änderung des Bildes verwendet.

Um dem Nutzer zu signalisieren, ob ein Tag ausgewählt wurde, wurde in *Home.vue* ein weiteres Attribut *selectedTag* angelegt. Jedem Tag-Element auf der Startseite wurde ein *OnClick Listener*, welcher bei einem Klick das jeweilige Element als *selectedTag* setzt hinzugefügt.

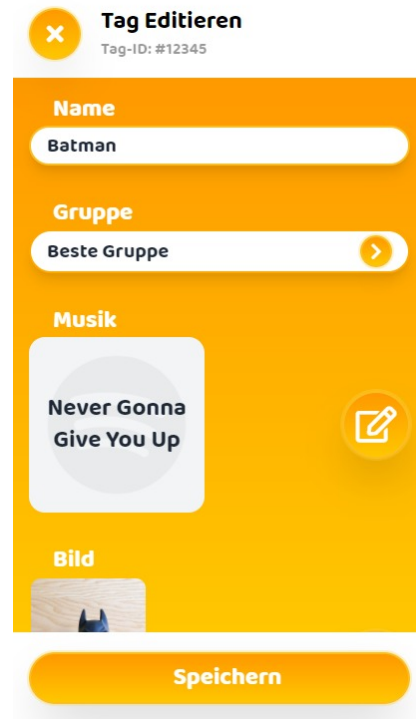


Abbildung 19: Tag Details



```

1      const toggleTag = (tag: NFCTag): void => {
2          selectedTag.value =
3              tag === selectedTag.value ? null : tag;
4          buttonTransitionActive.value = true;
5      };

```

Listing 7: Hervorgeben aus ausgewähltem NFC-Tag

Um dynamisch die CSS-Klasse der ausgewählten Tags zu ändern, wurde an dieser Stelle bei den *TagEntry*s der ternärer Operator genutzt. Je nachdem ob der *TagEntry* der gleiche wie der *selectedTag* war, wurden diesem andere Klasseneigenschaften gegeben.

```

1      <TagEntry
2          v-for="entry in group.tags"
3          :key="entry.nfcData"
4          class="m-4 w-2/6 flex-shrink-0 text-4xl"
5          :class="{ 'opacity-25': selectedTag !== entry &&
6                      selectedTag !== null
7                  }"
8          :img="entry.imageUrl"
9          :name="entry.name"
10         @click="toggleTag(entry)"
11     />

```

Listing 8: Caption

Wurde nun also ein Tag ausgewählt, wurden allen anderen die Klasse `opacity-25` zugewiesen, was einer Transparenz von 25% entsprach.

Im Sprint wurde auch das Produkt auf Auslieferbarkeit getestet, indem das komplette Setup durchgeführt wurde. Dabei wurde ein Setup-Guide<sup>48</sup> entwickelt, sodass andere Nutzer ihre eigene *leek-box* bei sich einrichten könnten.

Wie bereits beim vorherigen Meilenstein gestaltete sich auch hier das Zeitmanagement schwierig. Das Team hätte in diesem Sprint mehr Tickets umsetzen und dem Kunden präsentieren können, als zu Beginn versprochen. Jedoch fehlte die Zeit um die Features genügend zu testen und entsprechend fehlerfrei dem Kunden präsentieren zu können. Somit beschloss das Team dem Kunden im Review nur die anfänglich geplanten Features vorzustellen und die zusätzlichen Tickets im nächsten Sprint zu vollenden. Das Produkt konnte um die Einstellungen der *Leek Box* und die *TagDetails* erweitert werden. Auch der Setup Guide wurde fertiggestellt, sodass ein Nutzer selbstständig eine eigene *Leek Box* einrichten könnte. Bei dem Design wurden, bis auf ein paar Kleinigkeiten die im folgenden Milestone behoben wurden, alle

<sup>48</sup><https://github.com/project-leek/project-leek/blob/docs/Build.md>

Unstimmigkeiten beseitigt.

Das Team erhielt vom Kunden für das bisher entstandene Produkt viel Lob. Da das Team viel Gefallen an dem Projekt gefunden und entsprechend viel Energie aufgewendet hat, war dieses Lob sehr wertvoll und motivierend. Das Team konnte außerdem weitere Vue Funktionen kennenlernen. Problematisch war es, dass manche Tickets von mehreren Entwicklern zeitgleich oder nacheinander bearbeitet wurden, sodass einige der getätigten Änderungen redundant waren. Außerdem erfolgte die Kommunikation von größeren Änderung an jedes Teammitglied teilweise nur lückenhaft. Nach der Retrospektive wurde dies dem Team deutlich und es wurde sich auf eine bessere Kommunikation geeinigt. Lösungsvorschläge waren Verbesserung des Zeitmanagements durch Steigerung der Arbeitsstunden zu Beginn eines Sprints, sodass gegen Sprintende weniger offene Tickets vorhanden waren und das Team mehr Zeit zur Behebung möglicher Fehler vorhanden wäre. Eine bessere Kommunikation konnte durch das Festhalten von größeren Änderungen in den Pull Requests und Markierung aller Gruppenmitglieder erreicht werden.

## **6.6 Meilenstein 6**

Der sechste Meilenstein fand vom 28.01.2021 bis zum 15.02.2021 statt. Mit diesem Sprint sollte das User Interface finalisiert, alle Fehlerbehebung abgeschlossen und somit die Entwicklung beendet werden, sodass der Fokus komplett auf den Bericht gelegt werden konnte. So konnte der folgende und letzte Meilenstein für letzte Korrekturen am Bericht und das Vorbereiten der Projektpräsentation genutzt werden.

## Entwicklung

Zu Beginn wurden die letzten Softwaretickets abgeschlossen. Vor allem am User Interface wurden noch letzte Änderungen durchgeführt. Auf allen Seiten wurde die Höhe des *Header* und *Footer* auf die gleiche Größe angepasst. Aus der Datenbank wurden die manuell generierten Einträge und die Methoden zum automatischen *Seeding* der NFC-Tags gelöscht. Auf der Startseite wurde dem User nun ein Hinweis angezeigt, falls dieser noch keine NFC-Tags angelegt hat. Hierfür wird beim Laden der NFC-Tags die Anzahl geprüft und im Falle von keinem vorhandenen Tags ein *div* mit einem *GIF* und eines Hinweistextes über konditionales Rendering per *v-if* angezeigt oder ausgeblendet.



Abbildung 20: Startseite ohne Tags

## Button Refactor

Wie im Meilenstein vier bereits festgestellt, musste der häufig verwendete Button sehr anpassbar sein und in folgenden Versionen existieren:

- nur Text
- Text mit Icon links vom Text
- Text mit Icon rechts vom Text
- ein Icon ohne Text

Um diese und andere Anforderungen zu erfüllen, stieg die Komplexität des Buttons, bedingt durch eine Vielzahl von übergebenen Parametern, stark. Weiterhin musste die Größe des Buttons anpassbar, aber trotzdem möglichst einheitlich über die einzelnen Views hinweg, sein. Die *component* wurde vom Team gründlich überarbeitet um redundanten Code zu vermeiden. Statt den bisher zwölf Parametern, konnten alle Anforderung auch mit acht Parametern erreicht werden.

An dieser Stelle soll verdeutlicht werden, wie sich die Komponente selbst um die Darstellung kümmert, nachdem der Inhalt übergeben wurde:

```

1      <button
2      class="flex relative items-center outline-none focus:outline
        -none rounded-full border-yellow-400 border-2"
3      :class="containerClasses"
4      :disabled="disabled"
5      :type="type"
6      @click="doClick"
7      >
8      <slot>
9          <span
10             v-if="icon && text"
11             class="text-white"
12             :class="[...iconClasses, iconRight ? 'mr-4' : 'ml-4']"
13         />
14         <span
15             v-if="icon && !text"
16             class="absolute top-1/2 left-1/2 transform -translate-x
                -1/2 -translate-y-1/2 text-white"
17             :class="iconClasses"
18         />
19         <div v-if="!text && icon" :class="'icon-size-${size}'" />
20         <span
21             v-if="text || (!text && !icon)"
22             class="text-white mx-auto"
23             :class="[...textClasses, iconRight ? 'pr-4' : 'pl-4']"
24             >{{ text || 'Absenden' }}</span>
25         >
26     </slot>
27 </button>

```

Listing 9: Codesnippets vom Button

Interessant sind die dynamischen Abfragen *v-if* mit denen geprüft wird, ob zum Beispiel ein Text und oder ein Icon vorhanden ist. Entsprechend dieses Ergebnisses wurden dem Objekt unterschiedliche CSS-Klassen zugewiesen, wodurch der Button seine Form verändert. Ein Button der nur ein Icon enthielt, sollte zum Beispiel vollständig rund und das darin Icon zentriert sein. Um die Größe des Buttons zu steuern, wurde sich dafür entschieden eine *property* zu nutzen. So konnte an die *component* das Attribut „size“ mit den Werten „xs“, „md“ oder „xl“ übergeben werden. Dadurch konnte ein Button für die Welcome-Page deutlich größer sein als ein Speichern Button in der App selbst, und trotzdem sichergestellt werden, dass nur eine Art von Buttons existiert.

Auch wurde die Desktop Ansicht optimiert:

Da unsere Anwendung für die mobile Ansicht optimiert wurde, wurde in der Desktop Ansicht die Anwendung als kleineres Fenster im Handy Format

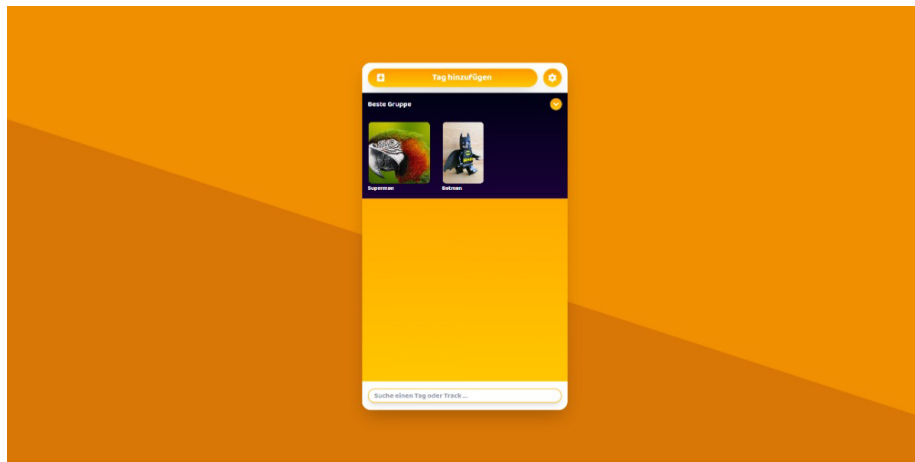


Abbildung 21: Desktopansicht

angezeigt. Dieses streckte sich auf höher aufgelösten Bildschirmen zu sehr in die Höhe in Relation zur Breite. Das wurde angepasst, dass die Relation Breite zu Höhe besser passt. Auch wurde ein schönerer Hintergrund hinter der eigentlichen Anwendung gesetzt, die auch farblich zur Farbpalette passt.

### Frontend auf Github.io Pages

Das Frontend ist vom Raspberry Pi zu Github.io Pages umgezogen. Somit kann der Endnutzer auch ohne Kenntnis der IP-Adresse des Rasperrys auf die Benutzeroberfläche zugreifen und muss keine weiteren Änderungen an der Netzwerkkonfiguration vornehmen, um von außerhalb des Heimnetzwerkes auf seine *Leek Box* zugreifen zu können. Die statischen Dateien von GitHub.io werden auf dem Endgerät geladen, sodass die Benutzeroberfläche angezeigt wird. In dieser kann der Nutzer die von *Ngrok* bereitgestellte Adresse seiner Leek-Box eingeben, woraufhin sich das Endgerät über einen verschlüsselten Tunnel mit dem Backend der Leek-Box verbindet. Anfänglich wurde diese Adresse jedoch nicht validiert, sodass die Eingabe einer fehlerhaften Adresse, in einem Fehler 404 resultierte. Bei einem Neustart des Rasperrys auch die ngrok-Instanz eine neue Adresse erhält, die anschließend manuell im *Spotify Developer Dashboard* eingetragen werden muss. Folglich musste der Nutzer nach jedem Neustart den Browsercache des Endgerätes löschen, um die Adresse der „neuen“ Leek-Box eingeben zu können.

### Einrichten einer Leek-Box

Die Einrichtung einer Leek-Box wurde in diesem Milestone beispielhaft durchgeführt und in einem Setup-Guide dokumentiert. Hierfür wurde auf einem neu aufgesetzten Raspberry Pi Docker installiert und das auf *DockerHub* verfügbare Backend-Image geladen. Zusätzlich wurde ein *Container* für den angeschlossenen USB-NFC-Reader aus der gleichen Quelle bezogen und gestartet. Dabei generiert der NFC-Reader eine ID, die im Backend registriert wird. Nach dem Start des Backend-Containers, gibt dieser auf der Konsole eine neue Adresse aus, die von *Ngrok*(vgl. 6.6) bereitgestellt wird. Daraufhin kann sich der Benutzer über `project-leek.github.io` durch die Eingabe dieser Adresse mit der neuen *Leek-Box* verbinden. Da in der App zu diesem Zeitpunkt noch kein Lautsprecher hinterlegt wurde, wird der Benutzer automatisch in die Einstellungen umgeleitet. Hier wird die neu angelegte *Leek-Box* zusammen mit der ID angezeigt und bietet dem Nutzer die Möglichkeit den Namen der Box festzulegen sowie einen vorhandenen Lautsprecher auszuwählen. Der zusätzliche Schritt der Auswahl einer Box wird benötigt, da die Lösung so konzipiert ist, dass ein Nutzer potentiell mehrere *Leek-Boxen* besitzen und die Verwaltung dennoch über ein zentrales Backend erfolgen kann. Somit entfällt die Einrichtung des Backends auf den einzelnen *Leek-Boxen* und muss lediglich einmalig durchgeführt werden.

### Bericht

Anfänglich wurden die Kapitel und Abschnitte zur Optimierung der Berichtsstruktur überarbeitet. Die daraus resultierenden Änderungen wurden in Tickets überführt, die sich die einzelnen Teammitglieder zuweisen konnten. Dies hatte den Vorteil dass, wie schon bei der Softwareentwicklung eine kontinuierliche Übersicht über den Fortschritt der Berichtsarbeit herrschte. Dadurch, dass das Team sehr viel Freude und Spaß am Entwickeln für das Projekt gefunden hatte, begann die dokumentarische Arbeit schleppend. Dies hatte zur Folge, dass zum Ende des Sprints nur etwas mehr als die Hälfte des

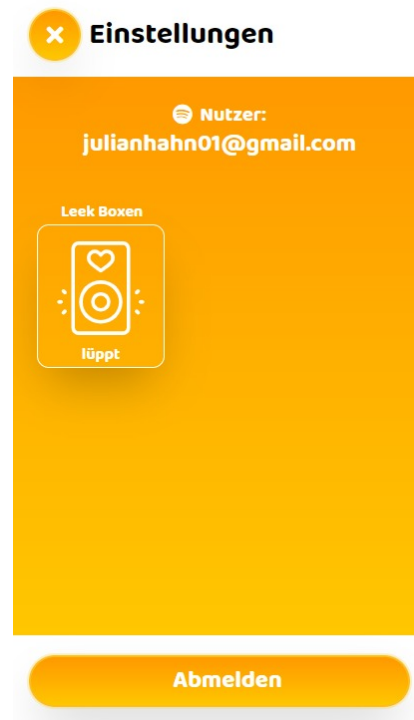


Abbildung 22: Einstellungen

Berichts abgeschlossen war. Somit konnte dem Prüfenden nicht die erste Version des Berichtes vorgelegt werden und das Team bekam somit verspätet erstes Feedback.

## Retrospektive

Dass der Bericht mit *LaTeX* geschrieben wurde hatte den Vorteil, dass Git sinnvoll genutzt werden konnte und die Änderungen der jeweiligen Mitglieder schnell ersichtlich waren. So konnte auch weiterhin der Workflow mit *Pull Requests* und zwei *Code Reviews* genutzt werden.

## 6.7 Meilenstein 7

Der finale Meilenstein wurde vom 15.02.2021 bis zum 05.03.2021 angesetzt.

## Letzte Bugfixes

Neben kleineren Bugfixes, wurde der Fehler behoben, der im letzten Review eine erfolgreiche Präsentation des initialen Verbindens mit der Box verhindert hatte. Dieser bestand darin, dass der Container des NFC-Readers versuchte, seine *Id* in diesem zu setzen, bevor er eine Verbindung zu dem Backend aufgebaut hatte. Dies konnte gelöst werden, indem ein *Event-Listener*, der die *Id* in der Datenbank des Backend setzt, auf das *connect*-Event registriert wurde. Ein weiteren Fehler trat auf, wenn keine Verbindung zur bereits gespeicherten Adresse der *Leek-Box* möglich ist. Dies trat zum Beispiel auf, wenn die *Leek-Box* nach einem Neustarte eine neue *Ngrok*-Adresse erhielt und resultierte in der Anzeige einer weißen Seite, ohne jegliche Fehlermeldung. Zur Behebung dieses Problems wurde eine Offline-Seite konzipiert, die Informationen über den Fehler bereitstellt und die Eingabe einer neuen Adresse ermöglicht.

## Bericht

Da die erste Version des Berichtes zu Beginn des Sprints noch nicht abgeschlossen war, musste das Team dies zeitig erledigen, um noch erste Anmerkungen vom Prüfenden erhalten und anschließend korrigieren zu können. Nach erneuter Rücksprache mit diesem

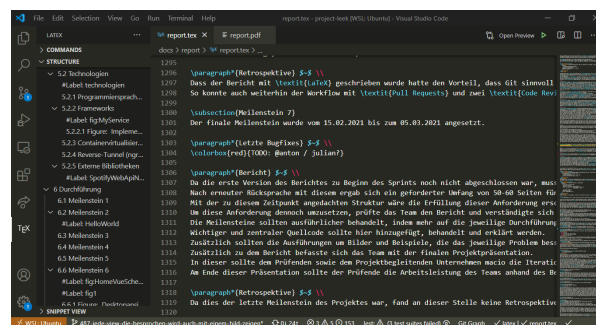


Abbildung 23: Bericht während der Erstellung

ergab sich ein geforderter Umfang von 50-60 Seiten für den Bericht (exklusive Anhang). Mit der zu diesem Zeitpunkt angedachten Struktur wäre die Erfüllung dieser Anforderung erschwert worden. Um diese Anforderung dennoch umzusetzen, prüfte das Team den Bericht und verständigte sich auf folgende Änderungen:

Die Meilensteine sollten ausführlicher behandelt, indem mehr auf die jeweilige Durchführung und Herangehensweisen eingegangen werden sollte; Wichtiger und zentraler Quellcode sollte hier hinzugefügt, behandelt und erklärt werden. Zusätzlich sollten die Ausführungen um Bilder und Beispiele, die das jeweilige Problem besser veranschaulichen ergänzt werden. Zusätzlich zu dem Bericht befasste sich das Team mit der finalen Projektpräsentation. In dieser sollte dem Prüfenden sowie dem Projektbegleitenden Unternehmen macio die Iterationen des Projektes und das Produkt präsentiert werden. Am Ende dieser Präsentation sollte der Prüfende die Arbeitsleistung des Teams anhand des Berichts und der Zufriedenheit des Kunden evaluieren können. Außerdem sollte der Kunde das Produkt selbstständig nutzen und weiterentwickeln können.

### **Retrospektive**

Da dies der letzte Meilenstein des Projektes war, fand an dieser Stelle keine Retrospektive statt.

## **7 Fazit**

### **7.1 Ausblick**

Am Ende der Bearbeitungszeit hat das Team ein Produkt geschaffen, welches die vom Kunden gestellten Mindestanforderungen erfüllt. Darüber hinaus gibt es noch einige weitere Aspekte der Produktvision, die noch umgesetzt werden könnten. Darunter befanden sich unter anderem eine Preisbeschränkung der *leek-box*-Hardware auf 30€ und die Möglichkeit eigene Lautsprecher zu nutzen. Auch die Benutzeroberfläche für die NFC-Tag-Konfiguration könnte noch weiter optimiert werden. Aber vor allem sollten andere Musikstreaming-Services neben Spotify eingebunden werden, welche dann in gleicher Weise zur Anmeldung und zum Hinzufügen und Abspielen von Musik genutzt werden. Nutzer könnten auch ihre Konten bei diesen Services verknüpfen und so NFC-Tags plattformübergreifend gemeinsam verwalten. Allerdings waren diese Vorstellungen von Anfang an optional, sie bildeten aber die allgemeine Produktvision und legten die Grundlage für mögliche Weiterentwicklung. Das Team hofft, dass Teile der Open-Source-Community genug Interesse an diesem Projekt haben werden, um daran weiter zu



entwickeln. Vielleicht haben diese auch weitere Ideen für die Zukunft der *leek-box*. Des weiteren besteht auch die Hoffnung, dass macio einiges von der Organisation dieses Projekts gelernt hat und auch fortlaufend weiter daraus lernen wird. So könnten dort in Zukunft auch weitere Projekte als Open-Source aufgestellt werden. Auch hofft das Team, dass auch in den nächsten Jahren des „Projekt Informatik“ mehr Open-Source Projektvorschläge gestellt werden. Dies würde Studierende damit konfrontieren Software zu erstellen, die nicht zwingend abgeschlossen wird. Sie würden auch lernen, Code für unbekannte Kollaborierende zu schreiben. Vor allem würde es aber zeigen, wie viel Spaß lebendige und langlebige angelegte Projekte machen können.

## 7.2 Projektreflexion

Das Projekt Informatik gab einen realistischen Einblick in die Arbeitswelt. Das Team musste sich klassischen Alltagsproblemen stellen, Lösungswege finden und konnte wichtige Erkenntnisse für die Zukunft daraus ziehen. Auch Coding-Entwurfsmuster und -Konventionen wurden neu erlernt und praktisch angewandt.

Zu Beginn des Projektes bestand eine größere Differenz, inwieweit die einzelnen Teammitglieder mit den jeweiligen Technologien vertraut waren. Insbesondere die Projektstruktur bereitete eine größere Einstiegshürde für viele Entwickler. Sie wurde durch das automatische generieren von Dateien und Ordern immer komplexer. Mit zunehmender Entwicklungszeit und durch die Arbeit an den Teilkomponenten des Mono-Repositorys gewannen die Entwickler jedoch schnell mehr Überblick über die Struktur. Das Team lernte wie Frameworks und Libraries z. B. Vue, Tailwind und Feathers sinnvoll und hilfreich eingesetzt werden können.

Zwar wurde den Studierenden im Studium der Vorteil von Versionskontrollsystemen wie Git vermittelt, doch fand sich selten ein praktisches Anwendungsszenario, dass Git über die Hauptfunktion der Versionierung hinaus sinnvoll genutzt hat. Dieses Projekt bot erstmals eine sinnvolle Anwendung von *git*. Die Umsetzungen der gängigen Industriestandards wie z.B. Pull-Requests und Code Reviews lehrte uns die Vorteile von Git in einer komplexen Umgebung. Diese Methoden führten zu folgenden maßgeblichen Verbesserungen: Mit zwei erforderlichen Code Reviews pro Pull Request, konnten Fehler im Code häufiger entdeckt werden und so eine höhere Codequalität erreicht. Als positiver Nebeneffekt konnten die Reviewer außerdem während des Prozesses neue Funktionen und Codemuster erlernen, zu denen sie bei Bedarf (per Kommentar) Fragen stellen konnten. Die Gewöhnung an diesen Deployment Cycle brauchte eine gewisse Zeit, fand aber

von Sprint zu Sprint immer mehr Wertschätzung im Team.

Jeder Pull Request durchlief mehrere automatisierte Tests - sogenannte *Actions* - die den geänderten Code auf Korrektheit in Syntax und simplen Semantischen fragen getestet haben. Inhaltliche Funktionen der Applikation wurden durch die Tests nicht abgedeckt. Diese prüften den Quellcode anhand fest eingestellter Testparameter. Durch diese Tests wurde dem Team viel manuelle Arbeit abgenommen und die Fehlerquelle Mensch minimiert. Änderungen im Master werden automatisch verarbeitet. Die neue Version des Codes wird auf Github.io veröffentlicht und geänderte Latex Dateien als PDF im eigenen Branch erstellt. Für Pull Requests wurde ebenfalls eine Vorschau erstellt und auf einer separaten Webseite veröffentlicht. Dank dieser Automatisierungen konnte sich das Team mehr auf das Weiterentwickeln des Projektes konzentrieren.

Durch die regelmäßigen Standups entstand ein Ort für den Austausch von Problemen und Informationen über Fortschritt und neue Funktionen. So wurde verhindert, dass Probleme untergingen, neues Wissen wurde schneller verbreitet und die gesamte Teamproduktivität verbessert. Das bei den Retrospektiven entstandene Feedback erwies sich als sehr wichtig für das Teamklima. Auch die hierbei angesprochenen Probleme und Verbesserungsvorschläge wurden vom Team als sehr wertvoll für die nächsten Sprints erkannt und entsprechend adaptiert.

Anfänglich hat sich das Team zu sehr auf die Standups, als alleiniges Kommunikationsmittel verlassen, wodurch die Absprache zwischen den Teammitgliedern nur selten stattgefunden hat. Der Umfang erster Tickets war deutlich zu groß. Die Komplexität eines Tickets hat Teammitglieder davon abgehalten sich mit der Thematik zu beschäftigen. Da konkrete Fragen bei einem gänzlich neuen Thema schwierig zu formulieren sind, wurde oft komplett auf Hilfe verzichtet und das Ticket zur Seite gelegt. Lösungsansätze wurden gemeinsam in den Retrospektiven erarbeitet. Weiterhin konnte offen über Probleme gesprochen werden, indem teaminterne Evaluationen stattgefunden haben.

Größere Probleme hatte das Team mit dem Zeitmanagement der einzelnen Tickets. Während eines Milestones wurden manchmal nicht alle geplanten Tickets fertiggestellt, oder es wurde nachträglich zu viel in den Sprint neu aufgenommen. Beides resultierte in Überstunden und zu wenig Zeit fürs Testen am Ende eines Meilensteins. Aus diesen Erfahrungen, hat das Team gelernt wie wichtig es ist die Planung des Meilensteins und dessen Tickets besonders sorgfältig zu gestalten. So hat das Team sich auch vorgenommen, mindestens zwei Tage vor einem Sprintende alle für den Sprint benötigten Tickets fertiggestellt zu haben. Dadurch sollte genügend Zeit geschaffen

werden, um Tickets auf Korrektheit zu testen und konfliktfrei in den Master Branch mergen zu können. Auch wenn die Lösung dieses Problem hätte effizienter behandelt werden können, war diese Methode angesichts der knappen Projektlaufzeit die effektivste.

## 8 Anhang

### Literatur

- [Bai+13] Engineer Bainomugisha u. a. „A Survey on Reactive Programming“. In: *ACM Comput. Surv.* (2013). URL: <https://doi.org/10.1145/2501654.2501666>.
- [Car20] Ryan Carniato. *JavaScript Frameworks, Performance Comparison 2020*. 21. Dez. 2020. URL: <https://medium.com/javascript-in-plain-english/javascript-frameworks-performance-comparison-2020-cd881ac21fce> (besucht am 11.02.2021).
- [Che18] Lianping Chen. „Microservices: Architecting for Continuous Delivery and DevOps“. In: März 2018. DOI: 10.1109/ICSA.2018.00013.
- [Kie21] FH Kiel. *Qualifikationsziele des Studiengangs Informationstechnologien*. 2021. URL: <https://www.fh-kiel.de/fachbereiche/informatik-und-elektrotechnik/studiengaenge/bachelor-studiengaenge/informationstechnologie-bachelor/qualifikationsziele/> (besucht am 08.02.2021).
- [Luc17] Aimee Lucido. *Monorepo to Multirepo and Back Again (Uber Technology Day)*. Apr. 2017. URL: <https://www.youtube.com/watch?v=1V8-1S28ycM> (besucht am 26.02.2021).
- [Sto20] Kathrin Stoll. *Tailwind – die CSS-Zukunft heißt Utility-First*. 28. Juli 2020. URL: <https://t3n.de/news/tailwind-css-zukunft-heisst-1299218/> (besucht am 22.02.2021).
- [Tor] Linus Torval. URL: <http://www.linux-kurs.eu/opensource.php> (besucht am 14.02.2021).
- [UK10] Malte Ubl und Eiji Kitamura. *Das Problem: Client-Server- und Server-Client-Verbindungen mit geringer Latenz*. 20. Okt. 2010. URL: <https://www.html5rocks.com/de/tutorials/websockets/basics/> (besucht am 22.02.2021).
- [Ulb17] Heinrich Ulbricht. *Drei Gründe warum TypeScript das bessere JavaScript ist*. 23. März 2017. URL: <https://www.communardo.de/blog/drei-gruende-warum-typescript-das-bessere-javascript-ist/> (besucht am 12.02.2021).