

# Projektbericht Leek-Box

Eine smarte Musik-Box

Projekt Informatik

Anton Bracke

Jan Eberlein

Tom Calvin Haak

Julian Hahn

Nick Loewecke

5. März 2021

# Inhaltsverzeichnis

<b>1 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 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 Machbarkeitsstudie</b>	<b>9</b>
3.1 NFC Tag . . . . .	9
3.2 Raspberry Pi . . . . .	10
3.3 User Interface . . . . .	10
3.4 Spotify . . . . .	11
3.5 Leek Box / Backend . . . . .	12
3.6 Andere Musikservices . . . . .	12
3.7 Sonstiges . . . . .	13
<b>4 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 . . . . .	15
4.1.3 Prototyping (Figma) . . . . .	16
4.1.4 Lucidchart . . . . .	17
4.2 Entwicklungszyklus . . . . .	17
4.3 Testing . . . . .	19
<b>5 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 Reverse-Tunnel (ngrok) . . . . .	29
5.2.4 Containervirtualisierung (Docker) . . . . .	29
5.2.5 Externe Bibliotheken . . . . .	30

5.3	Hardware	31
<b>6</b>	<b>Durchführung</b>	<b>31</b>
6.1	Meilenstein 1	31
6.2	Meilenstein 2	32
6.3	Meilenstein 3	36
6.4	Meilenstein 4	40
6.5	Meilenstein 5	46
6.6	Meilenstein 6	48
6.7	Meilenstein 7	53
<b>7</b>	<b>Fazit</b>	<b>54</b>
7.1	Ausblick	54
7.2	Projektreflexion	54
<b>A</b>	<b>Anhang</b>	<b>56</b>
A.1	Literatur	56
A.2	Projektskizze	58
A.3	User-Stories und Use-Flows	59
A.4	Mockups	68

# 1 Einleitung

Ein wesentlicher Bestandteil des Studiengangs „Informationstechnologie“ ist es, Softwareentwicklung in Teams und Kommunikation mit Kund:innen zu erlernen. [Kie21] Dies geschieht primär im Modul „Projekt Informatik (PROI)“, in welchem die Studierenden ein Semester lang in kleinen Gruppen an verschiedenen Projekten arbeiten, die häufig reale Sachverhalte der Softwareentwicklung umfassen und meist von Partnerunternehmen gestellt werden. Die Teams wählen wählen ihre favorisierten Projekte aus und stellen sich mit einer Kurzbewerbung bei den entsprechenden Unternehmen vor. Die Unternehmen entscheiden, welches Team ihr Projekt umsetzen darf.

Der nachfolgende Bericht dokumentiert den Verlauf und die Umsetzung des Projekts, welches das Autoren-Team für macio GmbH<sup>1</sup> aus Kiel realisierte.

## 1.1 Unternehmen

als einziges Unternehmen stellte macio in ihrer Ausschreibung die Anforderung, dass Projekt quelloffen (*Open Source*) umzusetzen. Das Team schätzte diese Bereitschaft, die geleistete Arbeitszeit nicht nur für den Eigenbedarf zu nutzen, sondern einen Beitrag für die Open-Source-Community zu leisten. Denn

*allgemein nützliche Information [sollten] frei sein. 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 Designen und Entwickeln von Mensch-Maschine-Schnittstellen. Neben der Rolle des Kunden konnten die Ansprechpartner:innen von macio bei Bedarf ihre jahrelange Erfahrung 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 wollte macio ihr Portfolio im IoT-Bereich erweitern sowie ihren Empfangsraum im Standort Kiel verschönern. Hierfür sollte eine smarte Musik-Box gebaut werden. Da die aktuell am Markt erhältlichen Konkurrenzprodukte wenig Konfigurierbarkeit bieten, zum Beispiel nur der Lautsprecher des Geräts genutzt und kein anderer Lautsprecher ausgewählt

---

<sup>1</sup>im folgenden nur noch „macio“ genannt

werden kann, sollte eine Open-Source Alternative geschaffen werden, die Nutzer:innen dabei Freiheiten böte.

### 1.2.1 Produktvision

Mithilfe der von macio gestellten Projektskizze (siehe Unterabschnitt A.2) erstellte das Team die folgende Produktvision: Die Konzeption und Entwicklung einer Musik-Box, die NFC Tags lesen kann und Schnittstellen zu Musik-Streaming-Anbietern (wie Spotify) besitzt. Würde auf die Box beispielsweise eine Figur mit integriertem NFC-Chip gestellt, sollte ein frei konfigurierbares Musikstück über eine bereits bestehende Musik-Anlage abgespielt werden. Falls es im Rahmen des Projektes möglich wäre, sollten die Nutzer:innen in der Lage sein, zwischen verschiedenen Musikanbietern zu wechseln. NFC-Chips und die zugehörige Musik sollten über ein Web-basierte Benutzeroberfläche konfiguriert werden können. Diese Benutzeroberfläche sollte von der Box ausgeliefert und primär für Smartphone-Bedienung gestaltet werden. Da es sich um ein Open Source Projekt mit entsprechender Lizenz handelte, müsste auch eine aussagekräftige, öffentliche Dokumentation verfasst werden. Macio würde die benötigte Hardware zur Verfügung stellen und bei technischen Fragen unterstützen.

### 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 funktional

Tabelle 1: Mindestanforderungen

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

Anforderung	Klassifikation
Sound Wiedergabe auf der Box selbst	funktional
Unterstützung anderer Musikdienste	funktional
3D-Modellierung und Druck einer passenden Box	nicht funktional
Maximale Kosten von 30€	nicht funktional
Cloud-Anbindung der Box	funktional
Auslieferung der Benutzeroberfläche 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 Musik-Box. Während die kostengünstige Variante nur die Grundfunktionen umsetzen würde, sollte die teurere einen größeren Funktionsumfang, wie das Abspielen der Musik über einen eingebauten Lautsprecher bieten. Einige der Anforderungen stellten sich während der Entwicklungsphase als widersprüchlich heraus. So entschied sich das Team zum Beispiel gegen bezahlte Services, was die Usability einschränkte, jedoch mehr dem Open-Source-Gedanken des Projekts entsprach. Ob und wie die Anforderungen umgesetzt werden konnten, wird im späteren Abschnitt Machbarkeitsstudie (Abschnitt 3) behandelt.

## 2 Projektplanung

### 2.1 Projekt Management

Zu Beginn des Projekts entschied sich das Team in Abstimmung mit macio für eine agile Projektorganisation. Diese Art des Projektmanagements zeichnet sich vor allem durch fortlaufende Produktentwicklung, kontinuierliches Feedback, kooperatives Arbeiten und Reaktionsfähigkeit bei Anforderungsänderungen aus. Diese Entscheidung wurde aus mehreren Gründen getroffen. Zum einen haben agile Methoden kein festes Endprodukt als Ziel, wie es beispielsweise bei klassischer Softwareentwicklung (Wasserfallmodell, V-Model, etc.) der Fall ist - 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 kürzere Veröffentlichungszyklen, die es dem Team ermöglichen, ein funktionierendes Produkt innerhalb eines Semesters<sup>2</sup> zu erstellen.

---

<sup>2</sup>Dieses Semester war aufgrund besonderer CoViD19-bedingten Auflagen einige Wochen kürzer als üblich.

Entsprechend dieser agilen Organisation wurde zu Beginn keine feste Planung des Projektverlaufs aufgestellt. Stattdessen entwickelte das Team eine Methodik um iterativ am Projekt zu arbeiten, wobei sich an der Methode Scrum[SS20] orientiert wurde. Von der Methode wurden vor allem die Einteilung in Projektiterationen fester Länge (Sprints) und die zugehörigen wiederkehrenden Meetings übernommen. Die Sprintlänge betrug im allgemeinen zwei Wochen<sup>3</sup>. Die Rollen der Teammitglieder von Scrum wurde nicht adaptiert, da sich im Team für eine Gleichverteilung der Aufgaben entschieden wurde.

## 2.2 Vor den Sprints

Am Anfang eines Sprints wurde ein Sprint-Planning durchgeführt, in dem das Team die Ziele für das Produktinkrement formulierte. Diese wurden gemeinsam mit dem Kunden abgesprochen und gegebenenfalls angepasst. Im Anschluss wurde das Produktinkrement des Teams anhand des geschätzten zeitlichen Arbeitsaufwands der einzelnen Änderungen in technisch orientierte Arbeitspakete aufgeteilt. Falls sich hierbei herausstellte, dass einzelne Arbeitspakete nicht umsetzbar oder zu aufwändig waren, wurde das Produktinkrement entsprechend angepasst.

## 2.3 Arbeit in den Sprints

Während jeden Sprints wurde das geplante Produktinkrement aus dem Sprint-Planing implementiert. Die einzelnen Arbeitspakete wurden, anhand der Anforderung und des Aufwands der jeweiligen Arbeitspakete, entweder alleine, in Paaren oder selten auch in Gruppen bearbeitet. Auch das individuelle 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 eingesetzt, um sowohl Flexibilität von Arbeitszeiten zu ermöglichen, als auch um die gewünschte Produktqualität sicherzustellen. Verschiedene Arbeitszeiten waren notwendig, da alle Teammitglieder verschiedene parallele Hochschulveranstaltungen besuchten und beruflichen Tätigkeiten nachgingen. So waren keine langfristigen synchronen Arbeitszeiten aller Mitglieder möglich. Die freie Verteilung von Paketen gestattete den Mitgliedern in persönlich bevorzugten Themengebieten arbeiten konnten, wodurch die Mo-

---

<sup>3</sup>Abgewichen wurde von dieser Länge nur zum Jahresende, um die Feiertage unterzubringen.

tivation des Teams am Projekt erhalten werden konnte.

Zur Abstimmung des Fortschritts während des Sprints, trafen sich die Teammitglieder drei Mal pro Woche (in der Regel montags, mittwochs und freitags). In Aufbau und Zweck orientierten sich diese regelmäßigen Meetings an den „Daily Standups“ von Scrum. Während der Meetings stellte jedes Teammitglied kurz vor was es seit dem letzten Standup am Projekt bearbeitet hatte, welche Probleme dabei aufgetreten waren, was bei der Arbeit gelernt wurde und welche Themen es bis zum nächsten Standup bearbeiten wollte. Nach diesen Kurzvorstellungen wurden einzelne besonders interessante und wichtige Punkte der Arbeit oder gelöste Probleme im Detail besprochen. So kam das Team auf den gleichen Wissensstand und konnte gemeinsam Entscheidungen treffen.

## 2.4 Am Ende der Sprints

Nach jedem Sprint wurde das Produktinkrement dem Kunden und der Projektbetreuung vorgestellt, sowie Ausmaß und Inhalt des nächsten Produktinkrements besprochen. Die resultierenden Wünsche und Rückmeldungen wurden ins Backlog übertragen. Im Anschluss fand eine Retrospektive statt, in der das Team die eigene Zusammenarbeit und den gemeinsamen Umgang evaluierte. Hierfür stellten sich alle Teammitglieder 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, fair und ehrlich. Die daraus entstandenen Erkenntnisse und Verbesserungsvorschläge wurden genutzt, um das Teamwork und die Arbeit im darauffolgenden Sprint weiter zu optimieren. Direkt nach jeder Retrospektive startete der nächste Sprint beginnend mit dem entsprechenden Sprint-Planning.

## 2.5 Quelloffenes Arbeiten

Teil der Anforderung war es, die Software als Open-Source-Projekt der Öffentlichkeit zur Verfügung zu stellen. Dementsprechend strebte das Team danach, das Projekt für andere Entwickler:innen erweiterbar und verständlich zu gestalten. Aufgrunddessen wurde der komplette Entwicklungsprozess auf *GitHub*<sup>4</sup> zu veröffentlicht. Die Plattform bot sich an, da sie für den Umfang des Projekts viele kostenlose Funktionalitäten in den Bereichen Automatisierung und

---

<sup>4</sup><https://github.com/>

Kollaboration bietet, welche halfen den Entwicklungsprozess intuitiv und zentral zu gestalten. Beispielsweise wurden die meisten fachlichen Diskussionen direkt in den Arbeitspaketen geführt, wodurch Entscheidungen und Denkprozesse von den Teammitgliedern und anderen Kollaborierenden nachvollzogen werden konnten. Dies wird auch bei der zukünftigen Instandhaltung und Weiterentwicklung des Projekts helfen.

Das Produkt-Backlog wurde in diesem GitHub Repository gepflegt, wodurch die Übersichtlichkeit des Projektstands erhöht wurde, da Arbeitspakete und zugehöriger Code sehr einfach verknüpft werden konnten. Die Möglichkeiten der Prozess-Automatisierung wurden genutzt, um den Workflow zu vereinfachen und um die Code-Qualität sicherzustellen. Dies geschah zum Beispiel durch die automatische Erstellung von Branches für einzelne Arbeitspakete bei Beginn der Bearbeitung und durch automatische statische sowie dynamische Tests des Codes.

Durch die Open-Source Anforderung konnte die Zielgruppe des Projekts auf Menschen mit fortgeschrittenen Technikkenntnissen festgelegt werden, was vor allem für das erstmalige Einrichten der *Leek-Box* und der zugehörigen Dokumentation maßgebend war. Entsprechend konnte technische Sprache bedient werden. Die Einschränkung der Zielgruppe auf technikaffine Benutzer:innen nicht für alle Breiche des Projekts. Die Benutzbarkeit der Nutzeroberfläche sollte einfach und ohne Vorkenntnisse möglich sein, damit die Musikbox zum Beispiel auch verschenk- oder für Familien nutzbar wäre.

### 3 Machbarkeitsstudie

Um Komplikationen während der Entwicklungsphase zu vermeiden, wurde vor der Durchführung eine Machbarkeitsstudie erstellt. Zu den einzelnen Anforderungen wurden technische Umsetzungen recherchiert und die jeweilige Umsetzung evaluiert.

#### 3.1 NFC Tag

##### Lesen und schreiben

Um NFC Tags zu nutzen, müssen mithilfe eines Hardware NFC-Readers Daten ausgelesen und an das Backend, den Raspberry Pi, übertragen werden. Dafür können Tastatureingaben, die die *Tag-IDs* darstellen, emuliert werden. *Evdev*, ein Kernel Modul von Linux, könnte zum Abgreifen dieser Eingaben genutzt werden.

Um neben der initial gespeicherten ID weitere Daten auf einen NFC-Tag schreiben zu können, benötigt der NFC-Tag einen internen Speicher. Ist dieser vor-

handen, können dort zum Beispiel Benutzerdaten hinterlegt werden. Dafür wäre ein spezieller NFC-Reader mit Schreibfunktionalität notwendig.

### 3.2 Raspberry Pi

#### Docker Integration

Docker Integration soll für einfachere Auslieferbarkeit angeboten werden (vgl. Unterunterabschnitt 5.2.4). Die Installation von Docker auf einem Raspberry Pi wird in vielen Anleitungen online beschrieben<sup>5</sup>, weshalb die Umsetzbarkeit dieses Punktes als sehr wahrscheinlich eingeschätzt wird.

#### Öffentlich zugängliches Web Interface

eineUm die Benutzeroberfläche der *Leek-Box* bereitzustellen, soll ein 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 ein DynDNS Dienst verwendet werden, welcher eine feste Domain in die wechselnde IP Adresse des Routers übersetzt. Alternativ könnte für das Projekt auch *Github.io Pages* genutzt werden. Dies würde bedeuten, 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.

### 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.

Um festzustellen, ob ein:e Benutzer:in bei einem Musik-Service angemeldet ist, muss sich dieser authentifizieren. Dabei muss in den meisten Fällen auch das Abomodell ermittelt werden, um die Möglichkeit der Nutzung von bestimmten Funktionen, wie der freien Wahl eines Songs, festzustellen. Dafür soll das offene Protokoll *OAuth* verwendet werden, welches von den häufig genutzten

---

<sup>5</sup><https://phoenixnap.com/kb/docker-on-raspberry-pi>

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 Identifikation anhand eines eindeutigen Merkmals sichergestellt werden. Die individuelle E-Mail Adresse wäre ein geeignetes Merkmal zur Wiedererkennung der Benutzer:innen. Über gesetzte *Scopes* in der OAuth Anfrage kann diese vom jeweiligen Provider mitgeliefert werden. Dafür müsste zum Beispiel vorrausgesetzt werden, dass ein:e Nutzer:in dieselbe E-Mail Adresse bei seinem:ihrem Musikstreamingdienst 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 Cover geladen und gespeichert werden. Beim primär verwendeten Musikstreaminganbieter Spotify sollte dies kein Problem darstellen, da zu jeder Anfrage von Titeln das Albumcover mitgeliefert wird.<sup>6</sup>

#### **Eigene Bilder nutzen**

Um Nutzer:innen weitere Personalisierung der angelegten Tags zu ermöglichen, könnten eigene Bilder für NFC-Tags verwendet werden. Eigene Bilder hochzuladen sollte generell möglich sein, doch ein einfacherer Weg wäre es, dass Nutzer:innen ein externes Bild verlinken. Sollten Nutzer:innen eigene Bilder hochladen wollen, würde dies mehr Aufwand bedeuten, da die Daten zum Backend gesendet und dort gespeichert werden müssten.

### **3.4 Spotify**

#### **Spotify Connect Lautsprecher auswählen und speichern**

Spezifische *Spotify Connect Lautsprecher* auszuwählen, 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

---

<sup>6</sup><https://stackoverflow.com/questions/10123804/retrieve-cover-artwork-using-spotify-api>

ein entsprechendes Lied zum Abspielen an den jeweiligen *Spotify Connect Lautsprecher* übergeben werden. Sollte nicht explizit ein Lautsprecher ausgewählt 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 könnten Nutzer:innen durch 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 von Nutzer:innen 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. Ein zentrales Cloud Backend hält eine Liste aller hinterlegten Verbindungsdaten bereit, sodass Nutzer:innen nur die *Id* einer *Leek-Box* anstelle einer gesamten Adresse eingeben müssen.

### 3.6 Andere Musikservices

#### **Unterstützung von Youtube Music**

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

#### **Unterstützung von Youtube**

Youtube bietet die Möglichkeit, nach Videos zu suchen<sup>8</sup>. Um diese auf dem Raspberry Pi als Musik abzuspielen, bietet sich *youtube-dl* zum Herunterladen der Videos als *.mp3* Dateien und der *omxplayer* zum Abspielen an. Hierfür wäre allerdings ein entsprechender Lautsprecher am Raspberry erforderlich. Ein vergleichbares System zu Spotify Connect existiert derzeit noch nicht. Auch die Umsetzbarkeit ist aufgrund des Herunterladens aus rechtlichen Gründen fraglich.

---

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

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

## **Unterstützung von Apple Music**

Der Anbieter Apple Music bietet mit MusicKit JS<sup>9</sup> eine Möglichkeit, 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 der Formfaktor einer physischen *Leek-Box* kompakt sein soll, müssten die Größe handelsüblicher 3D-Drucker ausreichend sein.

#### **Sound Wiedergabe auf internem Lautsprecher**

Manche Pi Modelle verfügen über einen On-Board Audio Anschluss, welcher aufgrund schlechter Auflösung qualitativ für Musik meist ungeeignet ist, weshalb ein weiteres Audiomodul oder eine externe Soundkarte verwendet werden sollte. Innerhalb der Raspberry Pi Reihe existieren in der Preisspanne von 20-30€ entsprechende Module.<sup>11</sup>. Zur Wiedergabe auf der Box selbst müsste dafür auf dem Raspberry eine Spotify Instanz ausgeführt werden, damit auch diese als *Spotify Connect Lautsprecher* erkannt wird. Hierfür existieren Bibliotheken wie *raspotify*<sup>12</sup>.

#### **Kostenpunkt unter 30€**

Für Raspberry Pi's ab der dritten Generation ist der festgesetzte Preisrahmen nicht mehr umsetzbar, da die UVP über der Preisbeschränkung 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>.

---

<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>

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

## 4 Tools und Vorgehen

### 4.1 Hilfsmittel

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

#### 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 *Hoster* für *Git-Remote-Repositories* einzusetzen. Aufgrund der bereits gewonnenen Erfahrung mit diesem Dienst und der Fokussierung auf quelloffene Software, stellte er für das Projekt die ideale Umgebung zur Quellcodeverwaltung und Kollaboration dar.

#### Kollaboration

Mit dem Anspruch das Produkt auch nach Projektende fortzuführen, wurden 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.

Eine weitere Funktionalität von GitHub, die in diesem Projekt genutzt wurde, sind die sogenannten *Actions*<sup>17</sup>. Sie ermöglichen automatisierte Tests, die den Quellcode und die Anwendung als Gesamtkonstrukt auf verschiedene Fehler prüfen und frei konfigurierbar sind.

---

<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/features/actions>

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>18</sup>**: Ist der Quellcode in gutem Stil geschrieben?
3. **test**: Sind die geschriebenen Unit-Tests erfolgreich?
4. **typecheck**: Sind alle Typen kompatibel?

Waren alle Tests erfolgreich, war die Änderung technisch nutzbar.

Auch wurden *Actions* genutzt, welche Tickets automatisch anhand von verschiedenen Events verschoben und einen Branch anlegten, wenn sich ein Entwickler einem Ticket zuordnete.

Außerdem wurde die in GitHub existierende Funktionalität der Pull Requests verwendet. Beim *Pushen*<sup>19</sup> werden die Änderungen automatisch in dem den Branch betreffenden Pull Request angezeigt (wenn vorhanden). Jeder Pull Request enthält allgemeine Informationen, welchem Zweck dieser dient und von wem die Änderung durchgeführt wird. Außerdem wird das Ergebnis der automatisierten Tests (*GitHub Actions*) angezeigt. Das Projekt wurde so konfiguriert, dass die Übernahme der Änderung des Pull Requests nur möglich war, wenn mindestens zwei andere Entwickler diesen Änderungen durch Code Reviews zustimmten und alle automatisierten Test erfolgreich abgeschlossen wurden. War dies der Fall, konnte die Änderung in den Master-Branch 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.

#### 4.1.2 Visual Studio Code & WSL

Als Entwicklungsumgebung (auch IDE) des Projekts wurde *Visual Studio Code* verwendet. *VSCodium* war allen Gruppenmitgliedern bereits bekannt und bereits

---

<sup>18</sup>Tool zur statischen Codeanalyse

<sup>19</sup>Hochladen der Änderung vom Entwickler-PC auf das Repository auf GitHub

regelmäßig verwendet. Die IDE beherrscht den Umgang mit allen für das Projekt benötigten Dateitypen und war durch eine Vielzahl an kostenlos angebotenen Erweiterungen (Extentions) 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 *VSCode* eine komfortable Anbindung<sup>20</sup> an das *Windows Subsystem für Linux (WSL)*<sup>21</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*.

#### 4.1.3 Prototyping (Figma)

Zur Abstimmung auf ein Design für die Benutzeroberfläche wurden vor der Entwicklung Prototypen der benötigten Komponenten mittels des webbasierten Prototyping-Tools *figma*<sup>22</sup> konzipiert. So konnten neben dem Design auch Abläufe in Form von „Click-Dummys“ erstellt und mit dem Kunden abgestimmt werden. *Figma* wurde verwendet, da es einen kollaborativen Zugriff ermöglicht, kostenlos ist und bereits Erfahrung mit der Plattform bestand.

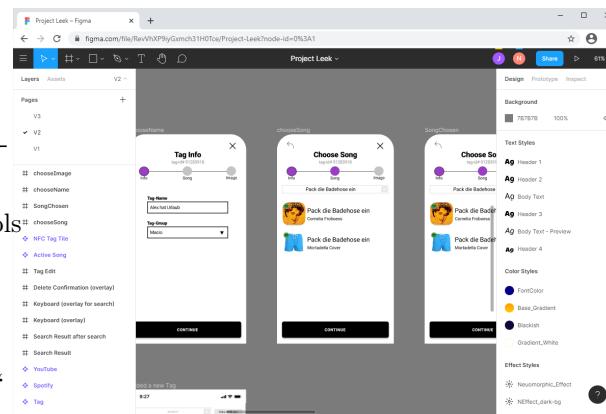


Abbildung 1: Prototypisierung mit *figma*

<sup>20</sup><https://code.visualstudio.com/docs/remote/wsl>

<sup>21</sup><https://docs.microsoft.com/de-de/windows/wsl/about>

<sup>22</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>23</sup> genutzt. Hier wurden neben der initialen Produktübersicht auch mehrere Aktivitätsdiagramme erstellt, die aus vom Projektteam entwickelten und mit dem Kunden abgestimmten User-Stories entstanden sind.

Diese sind im Anhang unter Unterabschnitt A.3 zu finden. Auch diese Plattform ermöglichte den gemeinsamen Zugriff und war den Entwicklern bereits bekannt, was die Notwendigkeit der Einarbeitung ausschloss.

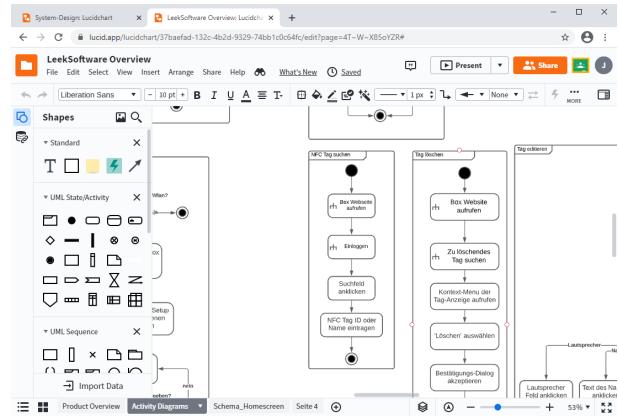


Abbildung 2: Zeichnen von Diagrammen mit *Lucidchart*

## 4.2 Entwicklungszyklus

Während der Entwicklung der *Leek-Box* durchlief jedes Teammitglied für jedes *Issue* den am Anfang des Projekts festgelegten und durch die Retrospektiven optimierten Zyklus, der folgendermaßen formuliert wurde:

Beginnend mit der Auswahl eines *Issues* aus der Spalte *To Do* des *Issue Boards*, weist sich der Entwickler nach Identifikation dem Ticket zu und daraufhin erstellt der verwendete Bot automatisch einen *Branch* in dem Schema *[Ticketnummer]-[Ticketname]*. Danach kann mit der Arbeit in *Visual Studio Code* begonnen werden.

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

Änderungen sollten möglichst kleinschrittig aber sinnvoll *committet* und anschließend *gepusht* werden. GitHub erkennt automatisch neue *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, setzt der Entwickler den Status des *Pull-Request* auf *Ready for Review*. Sieht ein Entwickler diesen *Pull-Request*, 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* getätigert werden. Ist der komplette Code geprüft, wird entschieden, ob die Änderungen angenommen (*approve*) werden oder eine Änderungsanforde-

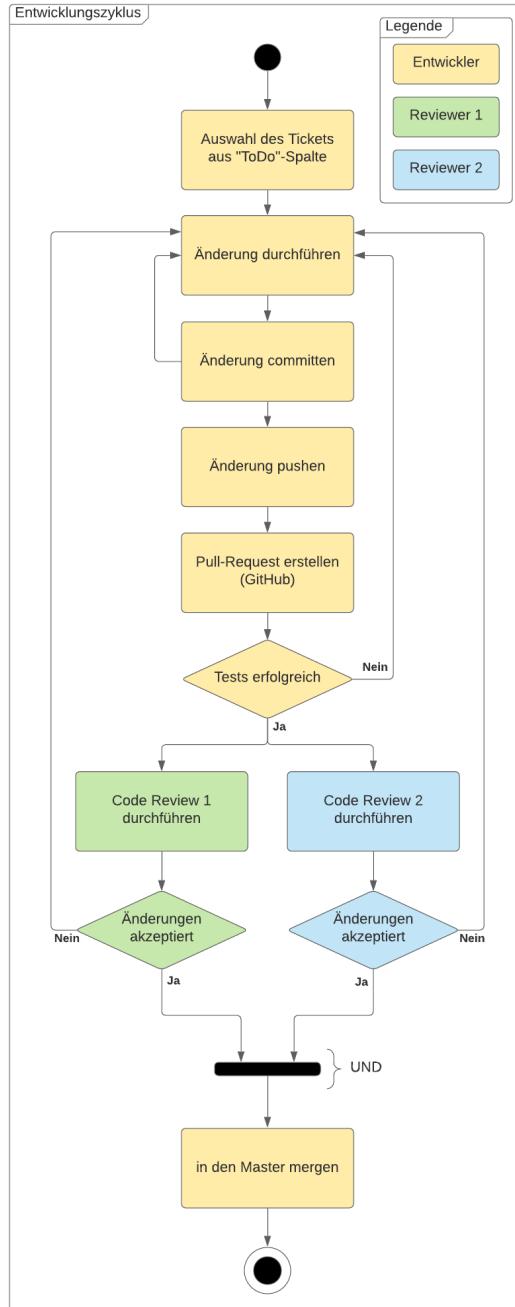


Abbildung 3: Entwicklungszyklus

rung gestellt wird (*Request changes*).

Im Falle von Änderungsanforderungen werden diese umgesetzt und ein erneutes Review angefordert. Dies geschieht zyklisch, bis die Änderungen approved werden. Sind die Änderungen durch zwei *Code-Reviews* bestätigt worden, werden sie in den Master Branch *merged*.

## 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 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 semantische (laufzeitunabhängige) Aspekte werden untersucht. So können kleinere Denkfehler und Flüchtigkeitsfehler, wie zum Beispiel fehlende Kommata, schnell gefunden und behoben werden. Hierfür wurde das Analyse-Werkzeug *ESLint*<sup>24</sup> genutzt. Für dieses Werkzeug war zusätzlich eine Erweiterung<sup>25</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>26</sup> genutzt. Diese stellte die selben Funktionalitäten für *Vue* 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 sukzessive 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 Teammitgliedern überprüft. Untersucht wurden vor allem semantische Fehler,

---

<sup>24</sup><https://eslint.org/>

<sup>25</sup><https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint>

<sup>26</sup><https://vuejs.github.io/vetur/>

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 Änderungsvorschläge gemacht und direkt übernommen werden, wodurch ein effizienter Review-Prozess gestaltet wurde.

### Dynamische Tests

Als dynamische Tests wurden in diesem Projekt Anwendungsfall-basierte Tests genutzt. Bei diesen Testfällen werden User-Storys 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 Entwicklungszyklus und wurden so in den meisten Fällen parallel zu den Reviews durchgeführt.

Ein gewisser Teil der anwendungsfallbasierten Tests fand auf der Ebene von Code-Funktionen (*Units*) statt, welche zur einfacheren Wiederholbarkeit automatisiert wurden. Dazu wurde das JavaScript-Test-Framework *Jest*<sup>27</sup> genutzt, da sich Tests hiermit sehr intuitiv und ohne großen Lernaufwand schreiben lassen. Auch diese Tests waren Teil des Entwicklungszyklus und wurden so bei jedem Pull-Request automatisch ausgeführt.

---

<sup>27</sup><https://jestjs.io/>

## 5 Technische Umsetzung

Auf Basis der Anforderung und der Machbarkeitsstudie evaluierte das Projektteam bekannte Technologien und recherchierte mögliche Alternativen für die Umsetzung der *Leek-Box*.

### 5.1 Projektstruktur

Die an das Projekt gesetzten Anforderungen machten die Konzeption 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 Microservice-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 in einem Repository zusammenzufassen. [Luc17] Repositorys dieser Art werden *Mono-Repositorys* oder *Monorepos* genannt, da sich alle Komponenten an einem einzigen Ort befinden. Jeder Service wurde in einem Unterordner in „*packages*“ angelegt (vgl. Abbildung 4). 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 Dokumentationsdateien wie Setup-Guide, Benutzerdokumentation und Projektbericht im Mono-Repo (im Ordner „*docs*“) abgelegt. Auch die für den Dienst *Docker* benötigten Dateien fanden in einem 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

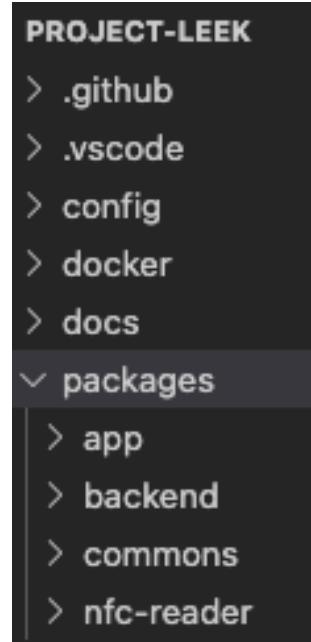


Abbildung 4:  
Ordnerstruktur

diesem Bericht verzichtet. Eine genaue Aufstellung ist dem *GitHub-Repository*<sup>28</sup> zu entnehmen. Durch die Zusammenfassung in ein Mono-Repository waren die einzelnen Services für jeden Entwickler jederzeit leicht erreichbar, ohne das Repository wechseln zu müssen.

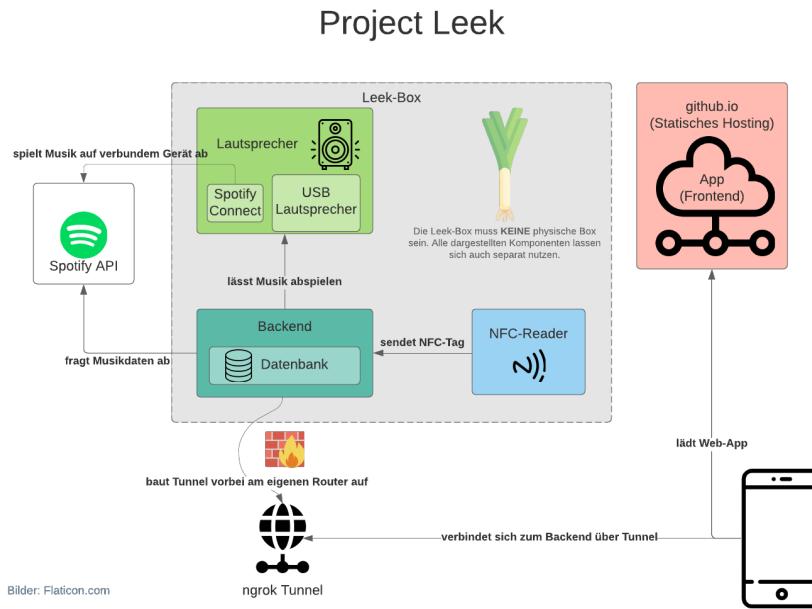


Abbildung 5: Überblick zum Aufbau der Leek-Software

## 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 einigten sich die Entwickler jedoch auf die von Microsoft entwickelte Sprache TypeScript. Diese bietet aufgrund des Aufbaus auf den ECMAScript-6-Standard eine große syntaktische Ähnlichkeit zu JavaScript, wodurch das Erlernen der Sprache weniger Zeit in Anspruch nahm. Durch die starke Typisierung entstehen außerdem wenige 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

<sup>28</sup><https://github.com/project-leek/project-leek>

JavaScript-Paketen zugegriffen werden konnte. Außerdem lässt sich TypeScript durch Nutzung der Laufzeitumgebung *Node.js*<sup>29</sup> auch serverseitig als Backend-sprache verwenden und ausführen 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 den Entwicklungsaufwand zu reduzieren.

#### 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 der Haupteigenschaften *components* und *reactivity* ausgewählt.

#### Components

Durch *Vue.js* 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 Teilespekte eines Elementes der Webseite in einer einzigen Datei.

---

<sup>29</sup><https://nodejs.org/en/>

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

```

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

Durch diese Zusammenfassung wird der gesamte Code lesbarer, da meist nur eine Datei je gerade behandeltem Element des Frontends konsultiert werden muss. So wurde den Entwicklern viel Suchzeit und Arbeitsaufwand erspart, die anderweitig angefallen wären, was zu effizienter und zufriedenstellender Arbeit am Projekt führte. Die gute Lesbarkeit führt langfristig zusätzlich zu besserer Wartbarkeit. Dies ist für das in diesem Projekt besonders wichtig, da es unter Open-Source-Lizenz steht und gegebenenfalls von der Allgemeinheit weiter entwickelt werden soll. Mit adäquater Wartbarkeit wurde so 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* intuitiv, wodurch das Framework eine flache Lernkurve hat. Die Verwendung von weitreichend klassischem HTML, CSS und JavaScript bzw. TypeScript 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, an-

hand welcher Inhalte, Aussehen und Verhalten dem Kontext entsprechend angepasst werden können. 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 Listing 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)">{{ name }}</button>
3   </template>
4
5   <script lang="ts">
6     import { defineComponent } from 'vue';
7
8     export default defineComponent({
9       name: 'Eintrag',
10      props: {
11        name: {
12          type: String,
13          default: 'Name',
14        },
15      },
16      emits: ['eintrag-geklickt'],
17      setup(props) {
18        const rückgabe: string = 'Eintrag ' + props.name + ', ausgewählt';
19        return { rückgabe };
20      },
21    });
22   </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 Eintragselement 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 reaktivier 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 müsste 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.js* 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 zusätzlicher Faktor für die Wahl des Frameworks *Vue.js* war die Performance. 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 nur begrenzt

---

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

te 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 Adapatern 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. Abbildung 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 dem 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 besserer Datenschutz für die Nutzenden gewährleistet werden.

### 5.2.3 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>35</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 *feathers* jedoch nativ kein HTTPS unterstützt und das Erstellen eines selbstsignierten Zertifikats sehr aufwändig ist, 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` auf. Dabei wird eine zufällige Subdomain im Schema `xyz.ngrok.io` angelegt und dem Benutzer in der Konsole des Backends angezeigt. Diese Adresse können die Benutzer:innen nun im Frontend (`https://project-leek.github.io`) eingeben, um sich mit der eigenen *Leek-Box* zu verbinden.

### 5.2.4 Containervirtualisierung (Docker)

Um die Installation der Box möglichst komfortabel zu gestalten, wird auf die freie Containervirtualisierungssoftware *Docker*<sup>36</sup> gesetzt. Ohne diese Möglich-

---

<sup>35</sup><https://ngrok.com/>

<sup>36</sup><https://www.docker.com/>

keit 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. Sie 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.

### 5.2.5 Externe Bibliotheken

#### Spotify Web API

Um verfügbare Songs eines Musikstreaming-Dienstes zu ermitteln und abzuspielen ist ein Zugriff auf dessen API notwendig. Der vom Kunden vorgeschlagene Anbieter war aufgrund hoher Nutzerzahlen *Spotify*<sup>37</sup>. Um den Zugriff auf die Spotify API zu simplifizieren wurde die Bibliothek *Spotify Web API Node*<sup>38</sup> verwendet, da diese die benötigten Methoden (z. B. die zum Suchen eines Songs anhand eines Suchbegriffs) zur Verfügung stellt. 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 per *Spotify-Connect* 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. *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.

---

<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>

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

### 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 bei der Wahl der Hardware durchgesetzt. Hierbei fungierte der Pi als Hoster des Backends und diente zur 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, ohne Spannungsquelle nutzbar Chips, die von einem NFC-Reader ausgelesen werden können. Die Daten können je nach NFC-Chip von nur einer *Id* bis hin zu beschreibbaren NFC-Tags mit unterschiedlichsten Daten variieren. In diesem Projekt wurden die primitiveren NFC-Tags, welche nur eine ID speichern, verwendet.



Abbildung 8:  
NFC-Reader und  
NFC-Tag

## 6 Durchführung

### 6.1 Meilenstein 1

Der erste Meilenstein erstreckte sich vom 29.10.2020 bis zum 11.11.2020 und war auf die Planung und Struktur des Projektes und Vorgehens fokussiert. Zum Ende des Meilensteins sollte die Struktur und das Grundgerüst für das Projekt fertiggestellt sein. Hierfür sollte ein Konzept der Software-Architektur sowie die genutzten Technologien (später Technologie-Stack genannt) auf Basis einer Anforderungsanalyse mit dem Kunden entwickelt werden. Um ein schnellen und reibungslosen Start für die Entwicklung zu gewährleisten,

sollte das Repository entsprechend vorbereitet und erste Tickets in das Backlog eingetragen werden. Das Team musste den eigenen Arbeitsablauf so konzipieren, dass jedes Teammitglied durchgehend sinnvoll arbeiten konnte. Durch bereits vorhandene Erfahrung mit verschiedenen Technologie-Stacks und ähnlichen Problemstellungen, wurde sich auf den unter Unterabschnitt 5.2 beschriebenen Technologie-Stack geeinigt. Um für jedes Teammitglied 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 aus 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. Das team entschied sich für den Produktnamen *Leek-Box* und trat entsprechend unter dem Namen *project-leek* auf. Für die Entwicklung wurde ein Mono-Repository aufgesetzt, welches Front- und Backend zusammenfasst. Durch die Anforderung der Veröffentlichung als Open-Source Projekt wurde eine verständliche und hilfreiche Dokumentation nötig, die Benutzer:innen 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.

## 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 zu 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 virtuelle Haustiere im Backend und die Darstellung derer im Frontend. Auch die Grundstruktur für den Bericht zu dem Projekt sollte in diesem Meilenstein erstellt werden, um Fortschritte, 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 Modell-Klasse erstellt werden, in der der Aufbau des Haustiers modelliert wurde. Sie beinhaltete Attribute wie „Name“ oder „Id“.

```

1   class 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: NicksPet (Teil von Hello World)

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 Modell-Klasse angegeben. Um z. B. Objekte der Klasse *NicksPet* zu verwalten wurde folgender Service angelegt:

```

1   class NicksPetService implements Service<NicksPet> {
2     [...]
3   }

```

Listing 6: NicksPetService (Teil von Hello World)

In dieser Klasse war außerdem ein Verweis aufgeführt, über welchen bestimmt wurde, mit welcher Schnittstelle oder API die Daten verwaltet werden sollten (in diesem Fall mit der durch *Feathers* bereitgestellten Schnittstelle für die Datenbank *NeDB*). Mit dem Erstellen dieser Klassen waren die benötigten Komponenten zur Verwaltung vorhanden und die Arbeit im Backend abgeschlossen. Der nächste Schritt befasste sich mit der 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 konnten.

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

### NFC-Tags auslesen

Neben der Umsetzung des Hello-Worlds sollte sich mit dem Raspberry Pi und

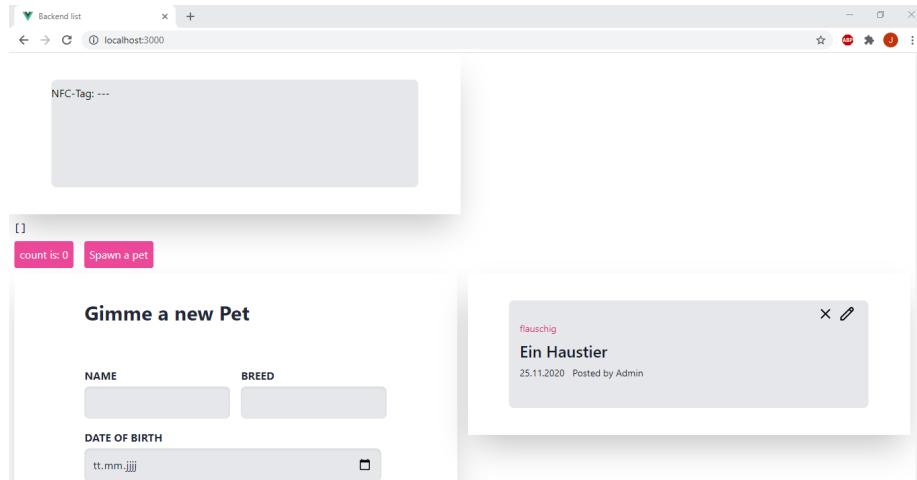


Abbildung 9: Zwei der Haustiere und die Anzeige des NFC-Readers

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, wurden 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 Abschnitt 6.2 beschriebenen Vorgehen erstellt. Außerdem wurde der Microservice *nfc-reader* in der Projektstruktur unter *packages* angelegt. Dieser nutzt die Bibliothek *Ev-dev*<sup>41</sup>, welche 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

<sup>41</sup><https://github.com/PixnBits/node-evdev>

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 Benutzeroberfläche**

Damit die Entwicklung zeitnah starten konnte, wurden die ersten Mockups für die Benutzeroberfläche erstellt. Mithilfe des Prototyping-Tools *figma* (vgl. Unterabschnitt 4.1.3) wurden Prototypen für die Start- und die Hauptansicht und die Ansichten für das Bearbeiten und Anlegen von NFC-Tags erstellt (Eine Übersicht über das erstellte Design ist im Anhang unter Unterabschnitt A.4 zu finden). Nach dem Design wurden die Ansichten in *Click-Dummies* 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, welche neuen Code auf etablierte Code-Konventionen und Lauffähigkeit überprüften.

### **Retrospektive**

Vor allem der Einsatz und das Know-How des Teammitglieds Anton wurde hier wertgeschätzt, welcher sich um die Automatisierung von Tests und Deployment kümmerte. Das Gruppenklima sowie die Zusammenarbeit, Kommunikation und die regelmäßigen Meetings wurden sehr gelobt. Das Team konnte bei der Umsetzung dieses Meilensteins viel über die Funktionsweise der Frameworks *React*, *Vue.js* und *Tailwind* lernen. Darüber hinaus kamen viele das erste mal mit automatisierten Development Pipelines, Code-Reviews und Pull Request in Kontakt. Das Team bemängelte bei dem Meilenstein jedoch auch, dass der Kundenwunsch zu spät behandelt und die Prioritäten falsch gesetzt wurden. Zur Lösung dessen wurde beschlossen die Sprintzielerreichung in den Standups häufiger zu kontrollieren.

### 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>42</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.

#### 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. Unterunterabschnitt 4.1.4) erstellt und sind im Anhang unter Unterabschnitt A.3 zu finden. Sie erleichterten die Umsetzung der Design-Mockups, indem sie notwendigen Abläufe visualisierten. Dank dieses Zwischen-schrittes 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 eine gute Usability für die Benutzer:innen sichergestellt werden. Außerdem konnten die *Flow-Charts* dem Kunden vorgelegt werden, um etwaige Miskommunikation bei der Anforderungsanalyse zu vermeiden und so die Wahrscheinlichkeit von späteren Änderungen wichtiger Funktion

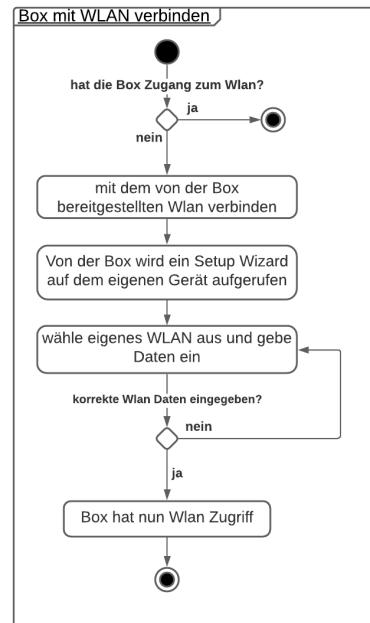


Abbildung 10: Flowchart: Connect to WIFI

<sup>42</sup>[https://moduldatenbank.fh-kiel.de/de-DE/Modules/Details/0de99d45-efc9-437a-a190-24d539b2a1d8](https://moduldatenbank.fh-kiel.de/de-DE/Module/Details/0de99d45-efc9-437a-a190-24d539b2a1d8)

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.

### **Musik abspielen**

Auch der Kundenwunsch, 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>43</sup> auseinander. Nach weiterführender Recherche entschieden die Entwickler die Bibliothek *Spotify Web API Node* (vgl. Abschnitt 5.2.5) zu verwenden, welche den Zugriff auf die *Spotify-API*, abstrahiert und so den Zugriff auf diese erleichtert. Die Bibliothek wurde in Kombination mit *Feathers-Hooks*<sup>44</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 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 besitzen muss. So konnten Funktionen wie das Anlegen eines neuen NFC-Tags von jedem Entwickler durchgeführt werden.

---

<sup>43</sup><https://developer.spotify.com/documentation/web-api/>

<sup>44</sup><https://docs.feathersjs.com/api/hooks.html>

## 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 wird von Spotify ein *JWT-Token* an das Backend übergeben. Ein *JWT-Token* besteht aus einem *Base64* encodiertem Text, welcher beliebige Daten beinhalten kann. 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 Accounts. Diese Daten werden extrahiert und sofern dieser nicht bereits in der Datenbank vorhanden ist, zum Anlegen eines neuen verwendet. Sollte dieser Account jedoch bereits in der Datenbank existieren, so wird dieser gegebenenfalls aktualisiert. Da sich jeder 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.

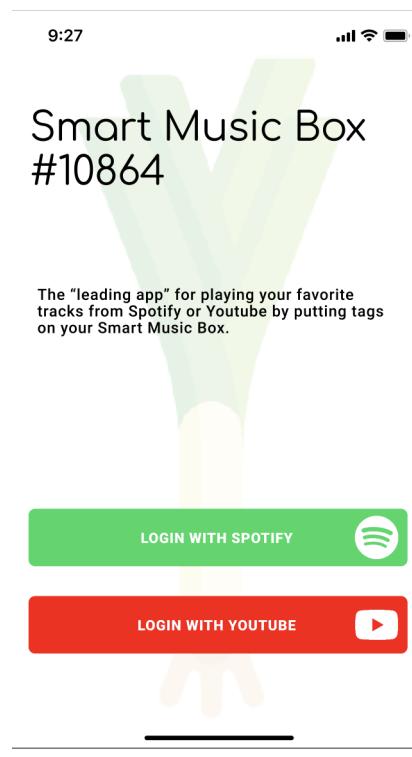


Abbildung 11: Welcome Screen  
Version 1

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*<sup>45</sup> so konfiguriert, dass sie nur aufgerufen wird, wenn die Benutzer:innen nicht angemeldet, also kein JWT-Token

<sup>45</sup>erlaubt das Wechseln von *Views* abhängig von der URL

vorhanden ist. Ist ein JWT-Token vorhanden, wird dem User bei dem versuchten Aufruf dieser Seite automatisch auf die Hauptseite weitergeleitet.

### **Veröffentlichung der App auf Github Pages**

Zur Veröffentlichung unseres Frontends der *Leek-Box* wird der Dienst Github Pages verwendet, 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 Continous-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 entstanden statischen Dateien von dem Pipeline Plugin *github-pages-deploy-action*<sup>46</sup> 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, 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**

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 durchgeföhrte Retrospektive ergab, dass die Problembewältigung durch die Anwendung von Pair Programming in diesem Sprint optimiert werden konnte. Auch der Workflow mit Github, Pull Requests und Code Reviews wurde verinnerlicht. Das Team lernte in diesem Sprint Vue Hooks und die Stärken von Feathers in Verbindung mit OAuth kennen. Festgestellt wurde, dass einige Tickets weiterhin stark voneinander abhängig waren, weshalb das Bearbeiten einiger Aufgaben nicht direkt möglich war. Da Abhängigkeiten teilweise unvermeidbar sind, wurde beschlossen, diese Tickets bei hoher Komplexität vermehrt mittels Pair-Programming zu bearbeiten, 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 Gelernetes besser kommuniziert werden, um die Arbeit effizienter zu gestalten. Hierfür wurden die *Standups* als Rahmen gewählt.

---

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

## 6.4 Meilenstein 4

Zu Beginn dieses Meilensteins waren alle Grundlagen geschaffen um mit der Umsetzung der Hauptfunktionen zu beginnen. Um über die gesamte Anwendung hinweg eingheitliches Design und Nutzbarkeit zu gewährleisten, sollten häufig verwendete Elemente entwickelt werden. Darüber hinaus 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

Außerdem sollten Mockups erstellt werden, die beschreiben, wie Nutzer:innen den Lautsprecher zur Wiedergabe auswählen sowie eigene *Leek-Boxen* initial einrichteten können. Aus diesen sollten später Views erstellt werden.

### Wiederverwendbare Steuerelemente

Für Steuerelemente, die häufig in der Anwendung genutzt werden (Button, Textfield, DropDownList), entwickelte das Team *components*, die in dem Ordner *uiBlocks* für alle Entwickler leicht zugänglich abgelegt wurden. Diese sollten garantieren, dass die Elemente anpassbar sind und dennoch einem einheitlichen Design folgen. Besonders anfangs halfen diese *components* den Entwicklern, da Design und Funktionalität lediglich zentral implementiert wurden und bei Bedarf in mehreren *views* jeweils einfach aufgerufen werden konnte, was den Arbeitsaufwand senkte. Bei einer solchen Einbindung musste dann nur noch der Inhalt über *properties* übergeben werden.

Während der fortschreitenden Entwicklung trat allerdings das Problem

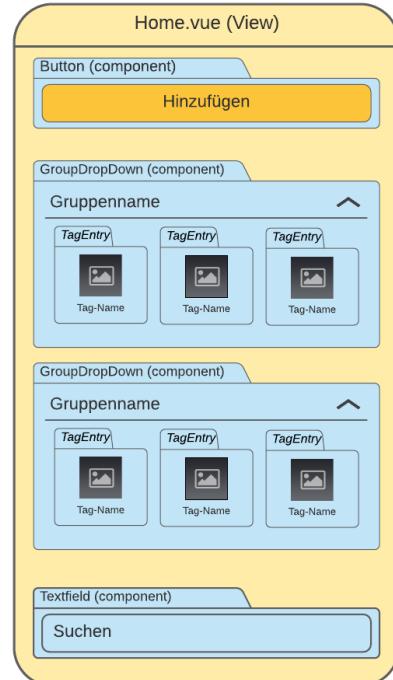


Abbildung 12: Schematischer Aufbau Home.vue aus *components*

auf, dass häufiger 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, wodurch die Übersichtlichkeit reduziert wurde. 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. Diese ist ein *accordion* und kann dementsprechend auf- und zugeklappt werden. Pro NFC-Tag in der jeweiligen Gruppe wird eine Instanz der *TagEntry component* in die Gruppe geladen.

Diese *components* werden in *Home* angezeigt. Neben der Einbindung von einer *GroupDropDown component* pro Gruppe wurde im Kopfbereich noch die vom Team erstelle *Button component* verwendet, die das Anlegen eines neuen NFC-Tags einleitet. Im Fußbereich befindet sich eine Suchleiste, die das Filtern der angezeigten NFC-Tags ermöglicht.

### Entwicklung der „Tag hinzufügen“ Funktionalität

Um dem Benutzer das Hinzufügen von neuen NFC-Tags zur 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. Im Hauptbereich der *View* werden die vier *components*, die für die Eingabe der NFC-Tag Daten zuständig sind, dynamisch geladen.

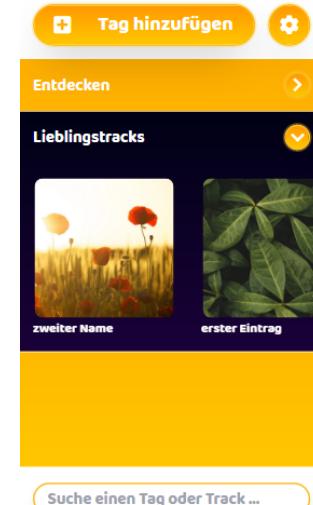


Abbildung 13: Home View

```

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

```

Listing 7: Dynamisches Laden der components von AddTag

Hierfür wird der beim Laden erstellte NFC-Tag (*newTag*) an alle *components* nacheinander weitergegeben und dort mit Informationen gefüllt. Der Fußbereich enthält abhängig von der im Hauptbereich gerade aktiven *componentButton 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 der sich in der Fußzeile befindliche Buttons abhängig vom der Gültigkeit verändert. So kann in jedem Schritt nur fortgefahren werden, wenn alle Eingaben valide sind. Einige dieser *components* werden ebenfalls für die Bearbeitung der NFC-Tags an anderer Stelle wiederverwendet.

Die Erste *component*, welche in den Hauptbereich geladen wird, ist *TagStepPlaceTagOnReader*. Diese 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 Events *update:nfc-tag* und *proceed* ausgelöst. *update:nfc-tag* bewirkt, dass das Attribut *TagId* des *newTag* gesetzt wird.



Abbildung 14: Step Scan  
(Add)

*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* für den Namen des NFC-Readers. Diese erhält eine *computed property*<sup>47</sup>, mit der sie im *setter* die Methode *updateTag* aufruft, 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 Add-Tag* den Wechsel zur nächsten *component* einleitet.

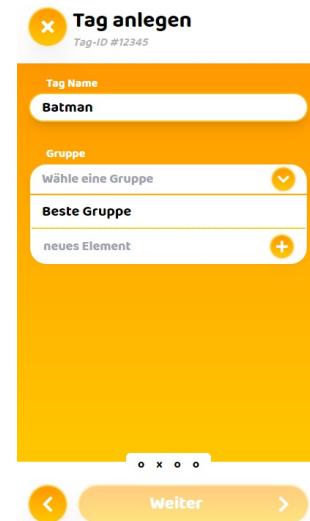


Abbildung 15: Add Step  
Scan-Tag

---

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

Hier in der *TagStepTrack* erhalten die Benutzer:innen die Möglichkeit, den Song auszuwählen, der beim Scannen des NFC-Tags abgespielt werden soll. Der gewünschte Song-Titel kann in eine *Textbox* zur Suche 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 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.

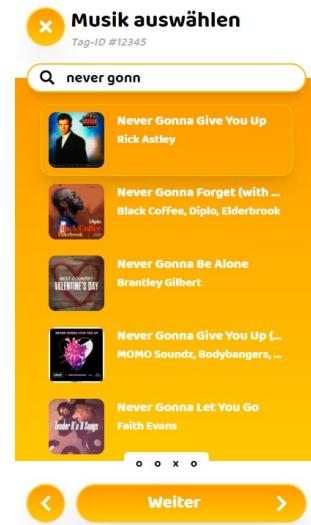


Abbildung 16: Add Step Track

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 den Benutzer:innen mehr Freiheiten in der Gestaltung zu geben, kann alternativ auch ein eigenes Bild verwendet werden. Dieses muss in der aktuellen Version auf einem Online-Dienst vorliegen. Ist dies gewünscht, kann in einer entsprechen-

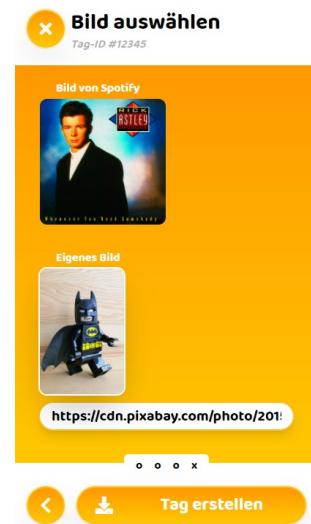


Abbildung 17: Add Step Image

den Textbox die *URL* dieses Bildes eingetragen werden, worauf das Bild automatisch zur Überprüfung durch die Benutzer:innen in ein dafür vorgesehenes Vorschaufeld geladen wird. Zusätzlich wird diese Textbox mittels einer *RegEx* 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.

### **Review & Retrospektive**

Durch die angestrebte Unabhängigkeit musste besonders viel Arbeit investiert werden, um die Features miteinander zu verbinden. Dieser Prozess dauerte länger als erwartet und forderte eine hohes Maß an Kommunikation zwischen den Teammitgliedern. Teilweise entstanden Wartezeiten und funktionierender Code wurde, über mehrere Merges, instabil oder defekt.

Nachdem die ersten großen Merges abgeschlossen waren, entschied sich das Team eine interne und anonyme Evaluation der Teammitglieder durchzuführen. Jedes Teammitglied erhielt somit konstruktive Kritik aber auch Lob und Unterstützung 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 war nun möglich, Tags anzulegen und entsprechend den Namen, Musik und ein Bild festzulegen und alle angelegten Tags zu durchsuchen. Auch wurde das von macio vorgeschlagene Design größtenteils umgesetzt.

Besonders gut gefiel den Entwicklern das Teamklima und das ehrliche Miteinander. Trotz aufgetretener Komplikationen und Problemen konnte dem Kunden ein funktionierendes Product Increment präsentiert werden. Pull Requests und Tickets wurden besser ausformuliert, 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 noch nicht optimal, da teilweise für einzelne Tickets zu viel Zeit aufgewendet wurde. Das Team stimmte überein, dass eine bessere Planung und Abstimmung für die nächsten Meilensteine notwendig wäre. Dementsprechend sollten granularere Tickets mit weniger Abhängigkeiten formuliert 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 Anwendungszenarien final umzusetzen, sodass die folgenden beiden Meilensteine zum Beheben kleinerer Fehler genutzt werden konnten. Somit war das Ziel je eine *View* für die Einstellungen der *Leek-Box* und die Bearbeitung von Tags umzusetzen. 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.

### Implementierung der Einstellungs-Ansicht

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:innen den Lautsprecher für die Wiedergabe auswählen. Außerdem enthält die Ansicht die Email-Adresse des aktiven Spotify-Accounts und einen Button zur Abmeldung.

### Bearbeiten von NFC-Tags

Auch die Funktionalität zum Bearbeiten der NFC-Tags wurde in diesem Meilenstein umgesetzt. An dieser Stelle zeigten sich erneut die Vorteile von *Vue*, da somit die bereits vorhandenen *components* wiederverwendet werden konnten. Um ein gleiches Design der Überschriften der einzelnen Steuerelemente zu garantieren, wurde die erstellte *component LabeledInput* für die Anzeige von Namen, Gruppe, Musik und Bild verwendet. Zur Änderung des Namens des Tags wird die 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 *nfc-*

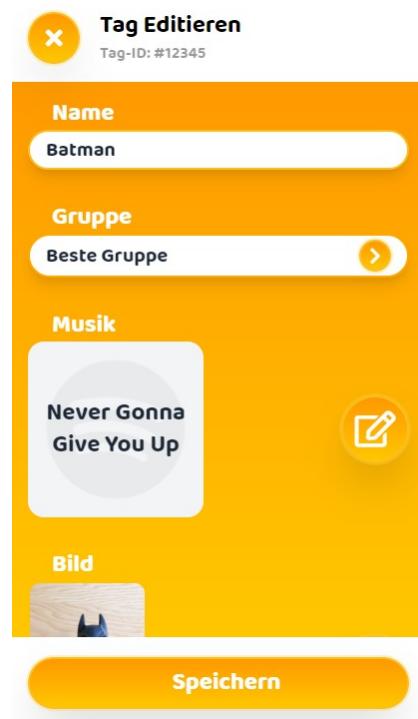


Abbildung 18: Tag Details

*Tag.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.

### Überarbeitung der *Home-View*

Um den Nutzer:innen zu signalisieren, ob und welches NFC-Tag ausgewählt ist, wurde die *Home.vue* um ein weiteres Attribut (*selectedTag*) erweitert. Jedem Tag-Element auf der Startseite wurde ein *onClick Listener*, welcher bei einem Klick das jeweilige Element als *selectedTag* setzt, hinzugefügt.

```

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

```

Listing 8: Hervorgeben aus ausgewähltem NFC-Tag

Ist ein Tag ausgewählt, wurden allen anderen die Klasse `opacity-25` zugewiesen, was einer Tranzparenz von 25% entsprach. Um die CSS-Klasse der ausgewählten Tags dynamisch zu ändern, wird überprüft, ob der *TagEntry* dem *selectedTag* entspricht.

```

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 9: Caption

### **Initiales Einrichten**

Um das Produkt auf Auslieferbarkeit zu testen, wurde der Installationsprozess auf einem zurückgesetztem Raspberry Pi durchgeführt. Dabei wurde ein Setup-Guide<sup>48</sup> entwickelt, sodass andere Nutzer:innen ihre eigene *Leek-Box* einrichten können.

### **Review & Retrospektive**

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 ausreichend zu testen und dem Kunden fehlerfrei 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 Nutzer:innen selbstständig eine eigene *Leek-Box* einrichten können.

Das Team erhielt vom Kunden für das bisher entstandene Produkt viel Lob, was für die Entwickler sehr wertvoll und motivierend war. Negativ angemerkt wurde, dass manche Tickets von mehreren Entwicklern zeitgleich oder nacheinander bearbeitet wurden, sodass einige der getätigten Änderungen redundant waren. Außerdem erfolgte die Kommunikation von größeren Änderungen an jedes Teammitglied teilweise nur lückenhaft. In 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 sein sollten und mehr Zeit zur Behebung möglicher Fehler verfügbar wäre. Eine bessere Kommunikation konnte durch das Dokumentieren und Präsentieren von größeren Änderungen und Markierung aller Gruppenmitglieder in den Pull Requests 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.

---

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

## Entwicklung

Zu Beginn wurden die letzten Software-tickets, die sich vor allem mit Änderungen am User-Interface befassten, abgeschlossen. Auf allen Seiten wurde die Höhe des *Headers* und *Footers* angeglichen.

Aus der Datenbank wurden die manuell generierten Einträge und die Methoden zum automatischen *Seeding* der NFC-Tags gelöscht. Auf der Startseite wurde den Nutzer:innen nun ein Hinweis angezeigt, falls diese noch keine NFC-Tags angelegt haben.

## 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 der Komponente, 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.

Folgend wird gezeigt, wie das Aussehen der Komponente über die *properties* gesteuert wird:



Abbildung 19: Startseite ohne Tags

```

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

```

Listing 10: 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 werden dem Objekt verschiedene CSS-Klassen zugewiesen, wodurch der Button seine Form verändert. Ein Button der nur ein Icon enthielt, sollte zum Beispiel vollständig rund und das enthaltene Icon zentriert sein. Um die Größe des Buttons zu steuern, wurde sich dafür entschieden eine *property* „size“ mit den möglichen Werten „xs“, „md“ oder „xl“ zu nutzen. Dadurch konnte ein Button für die *Welcome-View* deutlich größer dargestellt werden als ein „Speichern“ Button in der App selbst.

### Optimierung der Desktop-Ansicht

Da die Anwendung für mobile Geräte optimiert sein soll, wurde in der Desktop Ansicht die Anwendung als kleineres Fenster im Portrait-Format 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.

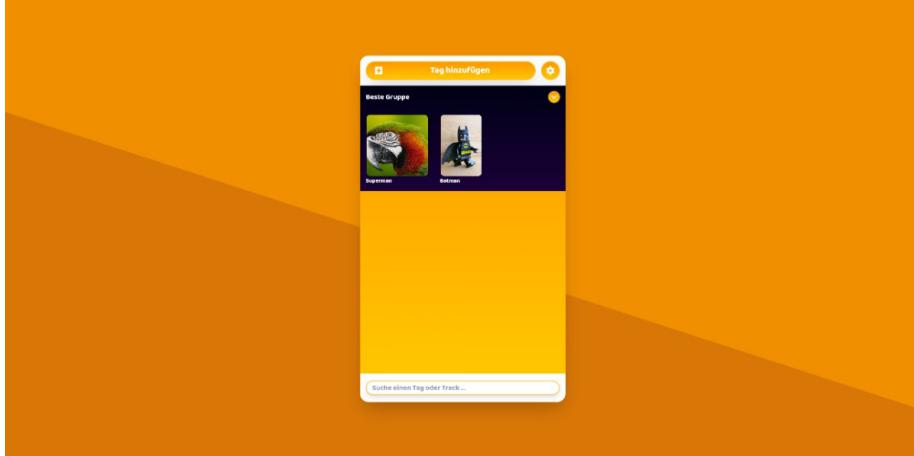


Abbildung 20: Desktopansicht

### Frontend auf Github.io Pages

Das Frontend ist vom Raspberry Pi zu Github.io Pages umgezogen. Somit können Nutzer:innen auch ohne Kenntnis der IP-Adresse des Raspberry Pis auf die Benutzeroberfläche zugreifen und müssen keine weiteren Änderungen an der Netzwerkkonfiguration vornehmen, um von außerhalb des Heimnetzwerkes auf die *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*(vgl. Unterunterabschnitt 5.2.3) 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 Raspberries erhält die ngrok-Instanz eine neue Adresse, die anschließend manuell im *Spotify Developer Dashboard* eingetragen werden muss. Folglich müssten die Nutzer:innen nach jedem Neustart den Browsecache 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 erneut beispielhaft durchgeführt, indem auf einem neu aufgesetzten Raspberry Pi Docker installiert und das auf *DockerHub* verfügbare Backend-Image geladen wurde. 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* bereitgestellt wird. Daraufhin können sich die Benutzer:innen 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, werden die Benutzer:innen automatisch in die Einstellungen umgeleitet. Hier wird die neu angelegte *Leek-Box* zusammen mit der ID angezeigt und bietet den Nutzer:innen die Möglichkeit, den Namen der Box festzulegen sowie einen vorhandenen Lautsprecher auszuwählen. Der zusätzliche Schritt der Auswahl einer Box wird benötigt, da das Produkt so konzipiert ist, dass Nutzer:innen potentiell mehrere *Leek-Boxen* besitzen können 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.

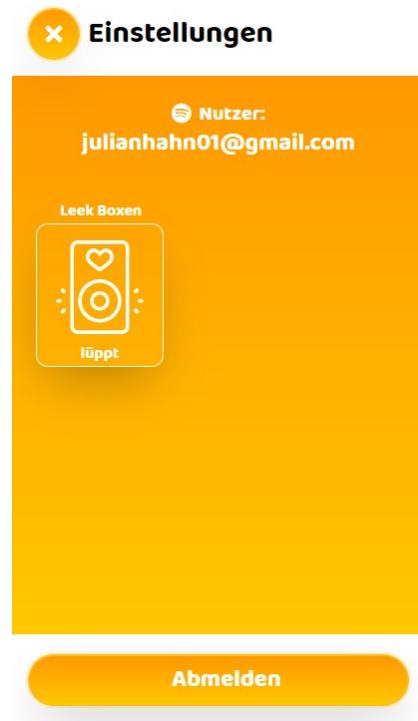


Abbildung 21: Einstellungen

### **Bericht**

Die Kapitel und Abschnitte des Projektberichts zur Optimierung der Struktur überarbeitet und die daraus resultierenden Änderungen wurden in Tickets überführt. Dies hatte den Vorteil dass, wie schon bei der Softwareentwicklung, eine kontinuierliche Übersicht über den Fortschritt der Berichtsarbeit herrschte. Aufgrund der starken Begeisterung an der Produktentwicklung, begann die Arbeit am Bericht schleppend. Infolgedessen war der Bericht zum Sprintende nicht wie geplant größtenteils abgeschlossen, sodass dem Prüfenden die erste Version des Berichtes nicht vorgelegt werden konnte und das Team somit verspätet erstes Feedback erhielt.

## Retrospektive

Die Wahl von *LaTeX* für den Bericht hatte den Vorteil, dass Git genutzt werden konnte, um kollaboratives Arbeiten zu ermöglichen und die Änderungen der jeweiligen Mitglieder schnell ersichtlich zu machen. Dies wäre in herkömmlichen Textbearbeitungs-Werkzeugen nicht möglich, da dort die Datei-Formate nicht textbasiert sind. So konnte auch weiterhin der Workflow mit *Pull Requests* und zwei *Code Reviews* genutzt werden.

## 6.7 Meilenstein 7

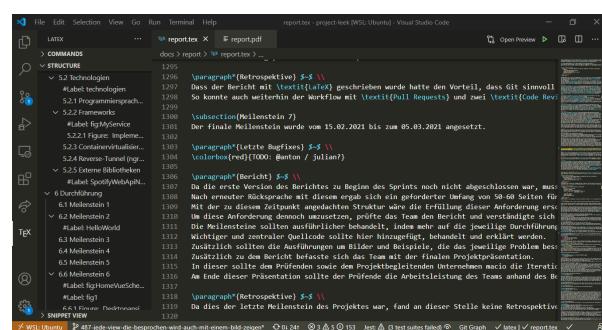
Der finale Meilenstein wurde vom 15.02.2021 bis zum 05.03.2021 angesetzt.

### 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 der Datenbank zu setzen, bevor er eine Verbindung zu dem Backend aufgebaut hatte. Der Fehler konnte gelöst werden, indem ein *Event-Listener* implementiert wurde, der die *Id* erst setzt nachdem das *connect*-Event ausgelöst wurde. Ein weiteren Fehler trat auf, wenn keine Verbindung zur bereits gespeicherten Adresse der *Leek-Box* möglich war. Dies war zum Beispiel der Fall, wenn die *Leek-Box* nach einem Neustart eine neue *Ngrok*-Adresse erhielt, was in der Anzeige einer weißen Seite ohne jegliche Fehlermeldung resultierte. 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 zeitnah erledigen, um erste Anmerkungen vom Prüfenden erhalten und anschließend korrigieren zu können. Nach erneuter Rücksprache mit diesem ergab sich ein ge-



The screenshot shows the Visual Studio Code interface with a LaTeX project open. The left sidebar shows a tree view of files and folders, including sections like 'STRUCTURE', 'COMMANDS', and 'REPORT'. The main editor area contains the LaTeX code for the report. The code includes sections for 'Retrospektive' and 'Meilenstein 7', and discusses the workflow and bugfixes mentioned in the text above. The status bar at the bottom indicates the file is saved and shows other standard VS Code icons.

Abbildung 22: Bericht während der Erstellung

forderter Umfang von 50-60 Seiten für den Bericht (exklusive Anhang). Mit der zu diesem Zeitpunkt angedachten Struktur wäre die Erfüllung dieser An-

forderung schwer umsetzbar gewesen. Um den geforderten Umfang dennoch zu erreichen, prüfte das Team den Bericht und verständigte sich auf folgende Änderungen:

Die Meilensteine sollten ausführlicher behandelt werden, indem mehr auf die jeweilige Durchführung und Herangehensweisen eingegangen werden sollte; Wichtiger und zentraler Quellcode sollte hier hinzugefügt, behandelt und erklärt werden. Außerdem sollten die Ausführungen um Bilder und Beispiele, die das jeweilige Problem besser veranschaulichen, ergänzt werden.

### **Abschlusspräsentation**

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.

## **7 Fazit**

### **7.1 Ausblick**

Am Ende der Bearbeitungszeit hat das Team ein Produkt geschaffen, welches nicht nur die vom Kunden gestellten Mindestanforderungen erfüllt, sondern auch einige der erweiterte Anforderungen umsetzt. Einige Aspekte der Produktvision könnten genutzt werden, um das Potential des Produktes noch weiter auszuschöpfen. Darunter befinden sich unter anderem eine Kostenreduzierung der *Leek-Box*-Hardware auf unter 30€ und die Möglichkeit eigene Lautsprecher zu nutzen. Außerdem könnte die Unterstützung anderer Musikstreaming-Services neben Spotify implementiert werden, um NFC-Tags plattformübergreifend zu verwalten. Das Team hofft, dass in der Open-source-Community ein breites Interesse an der Weiterentwicklung des Projekts entsteht. Weiterhin würden sich die Entwickler wünschen, dass in den nächsten Jahren des „Projekt Informatik“ mehr Open-Source Projektvorschläge gestellt werden.

### **7.2 Projektreflexion**

Zu Beginn des Projektes bestand zwischen den einzelnen Teammitgliedern eine größere Differenz in den Kenntnissen der Technologien. Insbesondere das Zusammenspiel der vielfältigen Technologien stellte eine große Einstiegshürde für die Entwickler des Teams dar. Durch die hohe Hilfsbereitschaft der einzelnen

Teammitgliedern und diversen Gelegenheiten zur Anwendung des erworbenen Wissens, konnten die Entwickler ihre Fähigkeiten im Bereich der Webentwicklung mit Frameworks wie *Vue.js*, *Feathers* und *Tailwind* ausbauen.

Zwar wurde den Studierenden der Vorteil von Versionskontrollsystmen wie *git* vermittelt, doch fand sich selten ein praktisches Anwendungszenario, das *git* über die Hauptfunktion hinaus sinnvoll genutzt hat. Dieses Projekt zeigte die Vorteile von *git* auf und bot erstmals eine sinnvolle Anwendung, in der die in modernen IT-Unternehmen genutzten Methoden wie z. B. Pull-Requests und Code Reviews genutzt wurden. Diese führten zu maßgeblichen Verbesserungen: Mit zwei erforderlichen Code Reviews pro Pull Request, konnten Fehler im Code häufiger entdeckt und so eine höhere Codequalität erreicht werden. 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. Nach einer anfänglichen Gewöhnungsphase schätzte das Team die aus dem Development-Cycle resultierende Stabilität des Produktes sehr.

Rückblickend hat sich die agile Projektorganisation als eine gute Entscheidung erwiesen. Die regelmäßigen Standups boten eine Möglichkeit für den Austausch von Problemen und Informationen, sowie der Kommunikation von Fortschritt und neuen Funktionen. So wurde verhindert, dass Probleme untergingen, neues Wissen wurde schneller verbreitet und die gesamte Teamproduktivität optimiert. Auch das bei den Retrospektiven entstandene Feedback erwies sich als wichtig für das Teamklima. Die hierbei angesprochenen Probleme und Verbesserungsvorschläge wurden vom Team als sehr wertvoll für die nächsten Sprints erkannt und entsprechend adaptiert.

Abschließend lässt sich sagen, dass das „Projekt Informatik“ einen realistischen Einblick in die Arbeitswelt ermöglichte, und erwies sich damit besonders nützlich für angehende Softwareentwickler:innen. Das Team musste sich klassischen Alltagsproblemen stellen, Lösungswege finden und konnte wichtige Erkenntnisse für die Zukunft ableiten. Auch Entwurfsmuster und Code-Konventionen wurden neu erlernt und sofort anhand praktischer Anwendung erprobt. Trotz einiger Schwierigkeiten während der Entwicklung, konnte durch kontinuierliche Evaluation des Vorgehens die Schwierigkeiten sukzessive minimiert und so das „Projekt Informatik“ zum erfolgreichen Abschluss geführt werden.

## A Anhang

### A.1 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-inplain-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-elekrotechnik/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).
- [SS20] Ken Schwaber und Jeff Sutherland. *The 2020 Scrum Guide*. 2020. URL: <https://scrumguides.org/scrum-guide.html> (besucht am 24.02.2021).
- [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).

## A.2 Projektskizze

### Smart Music Player mit NFC und Spotify Connect

Smarte Spielzeuge mit NFC-Funktion sind inzwischen keine Seltenheit mehr, aber sehr oft stehen dahinter große Spielzeugkonzerne, die für diese Produkte viel Geld verlangen. Wir möchten in diesem Projekt eine Open-Source-Alternative entwickeln, unser Portfolio im IoT-Bereich erweitern und gleichzeitig unseren Empfangsraum in Kiel verschönern.

Dazu soll eine Box bestehend aus einem Mikrocontroller mit NFC-Reader entworfen werden, die sich mit Musikdienst-APIs wie Spotify Connect verbinden kann. Auf diese Box können kleine Gegenstände (Würfel, Figuren) mit NFC-Tags gestellt werden, wodurch das Abspielen eines Songs, eines Podcasts oder einer Playlist auf einem verbundenen Gerät (z.B. Handy, Smart Speaker) ausgelöst wird. Die Konfiguration der entsprechenden Trigger erfolgt über ein Web-Frontend, das direkt von der Box ausgeliefert wird und auch mit einem Smartphone leicht zu bedienen ist.

Das Projekt umfasst die gesamte Entwicklung vom Auslesen der NFC-Tags über die Backend-Verbindung mit dem Musikdienst bis zum Konfigurations-Frontend. Die entwickelte Software soll unter einer Open Source Lizenz frei verfügbar gemacht werden, weshalb auch eine aussagekräftige, öffentliche Dokumentation verfasst werden muss.

Alle benötigte Hardware sowie einige Spotify-Accounts zum Testen werden zur Verfügung gestellt. Vorgehensmodell und Umsetzungstechnologien können vom Projektteam selbst bestimmt werden, wobei wir beratend und unterstützend zur Verfügung stehen.

### Unternehmensportrait macio

Die macio GmbH kreiert individuelle Bediensoftware mit Innovationscharakter. Im Auftrag namhafter Unternehmen aus der Industrie entwickeln wir professionelle, interaktive Anwendungen für den Geräte-, Maschinen- und Anlagenbau sowie für die Labor- und Medizintechnik.

#### Impulsgeber echter Innovationen

Innovatives Software Engineering und ausgezeichnetes User Interface Design – mit dieser fein abgestimmten Know-how-Kombination entwickeln wir Bedienerlebnisse für unterschiedliche Use Cases: direkt an der Maschine auf einem Embedded-Gerät, als mobile App inklusive Vernetzung der Geräte zur Beobachtung aus der Ferne oder als stationäre Desktopanwendung.

Ergebnisorientiert, prozessgenau und zertifiziert begleitet unser interdisziplinäres Team aus Technologie- und Designexpert\*innen das gemeinsame Projekt von der Spezifikation einer Produktidee über die Konzeption und Umsetzung bis hin zur Produktpflege im Markt.

Seit 2002 blicken wir auf bereits über 350 erfolgreich entwickelte Projekte zurück. An unseren Standorten in Kiel, Karlsruhe, Düsseldorf und Hamburg entwickeln wir mit unseren Kunden auf Augenhöhe innovative Lösungen, die einfach funktionieren.

### Ansprechpartner

Kieran Murtagh  
[kieran.murtagh@macio.de](mailto:kieran.murtagh@macio.de)

Stefan Krüger  
[stefan.krueger@macio.de](mailto:stefan.krueger@macio.de)

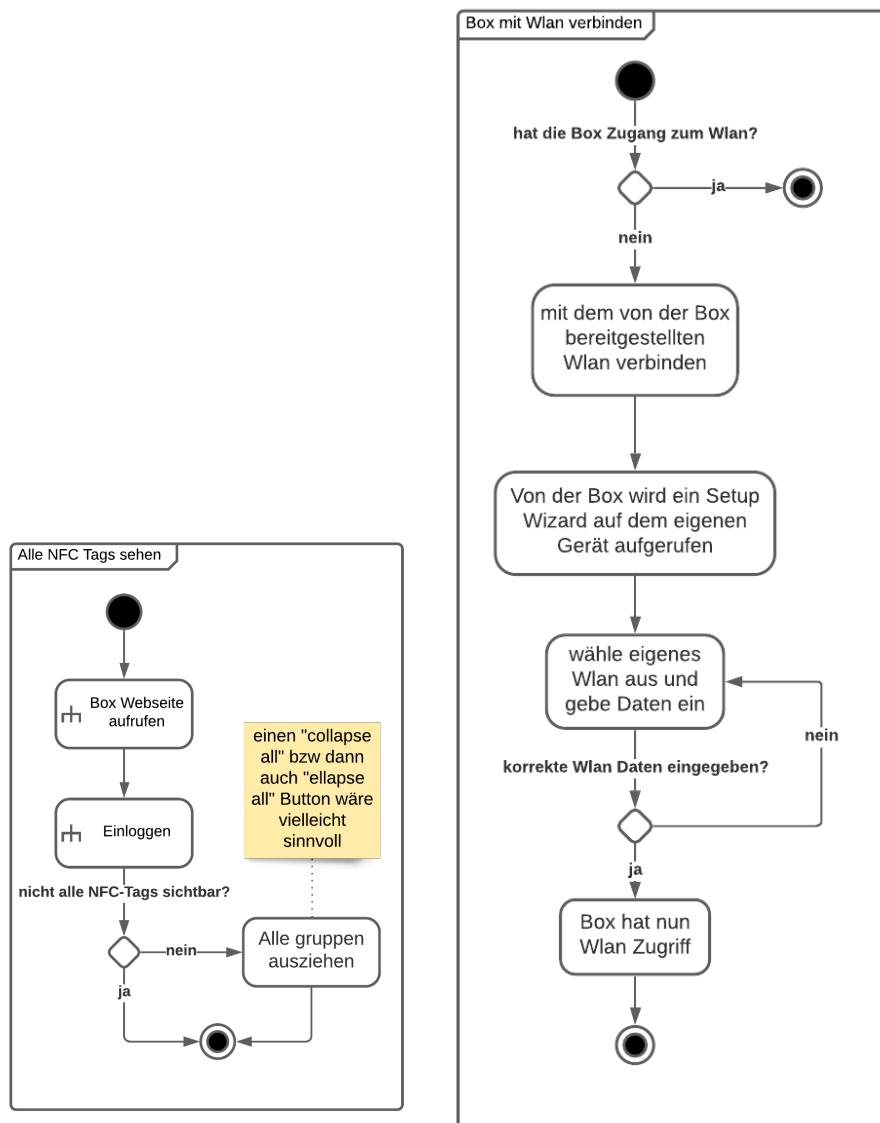
## A.3 User-Stories und Use-Flows

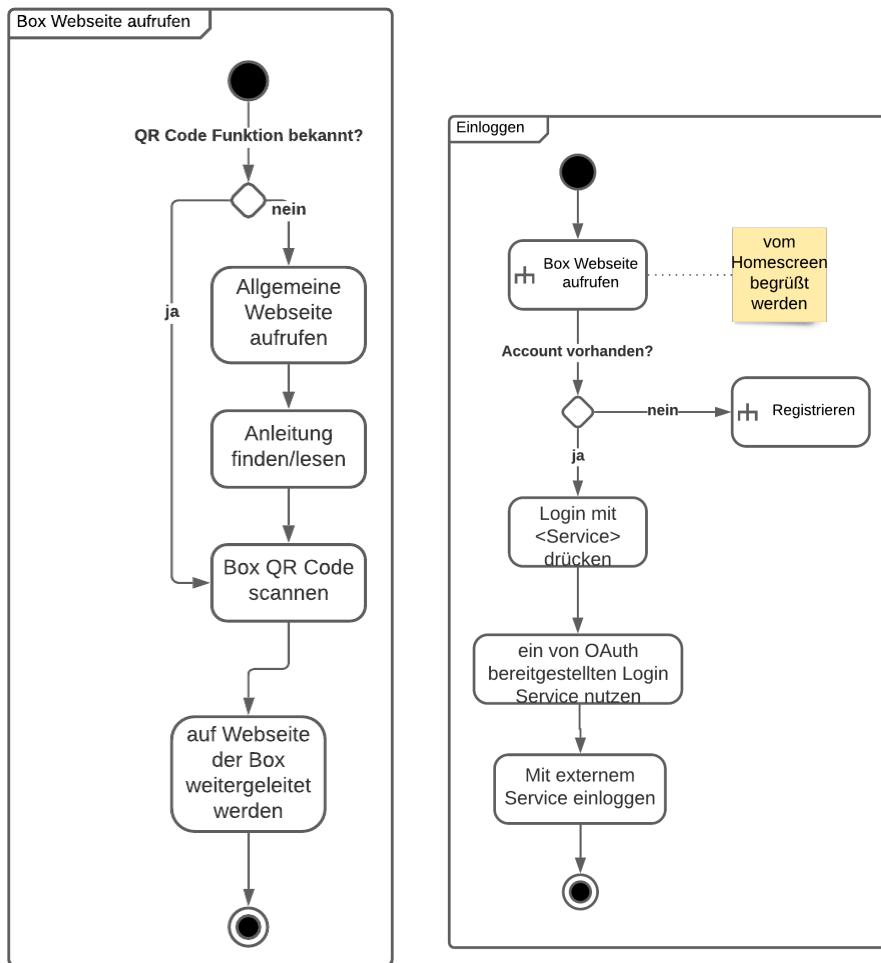
### User-Stories

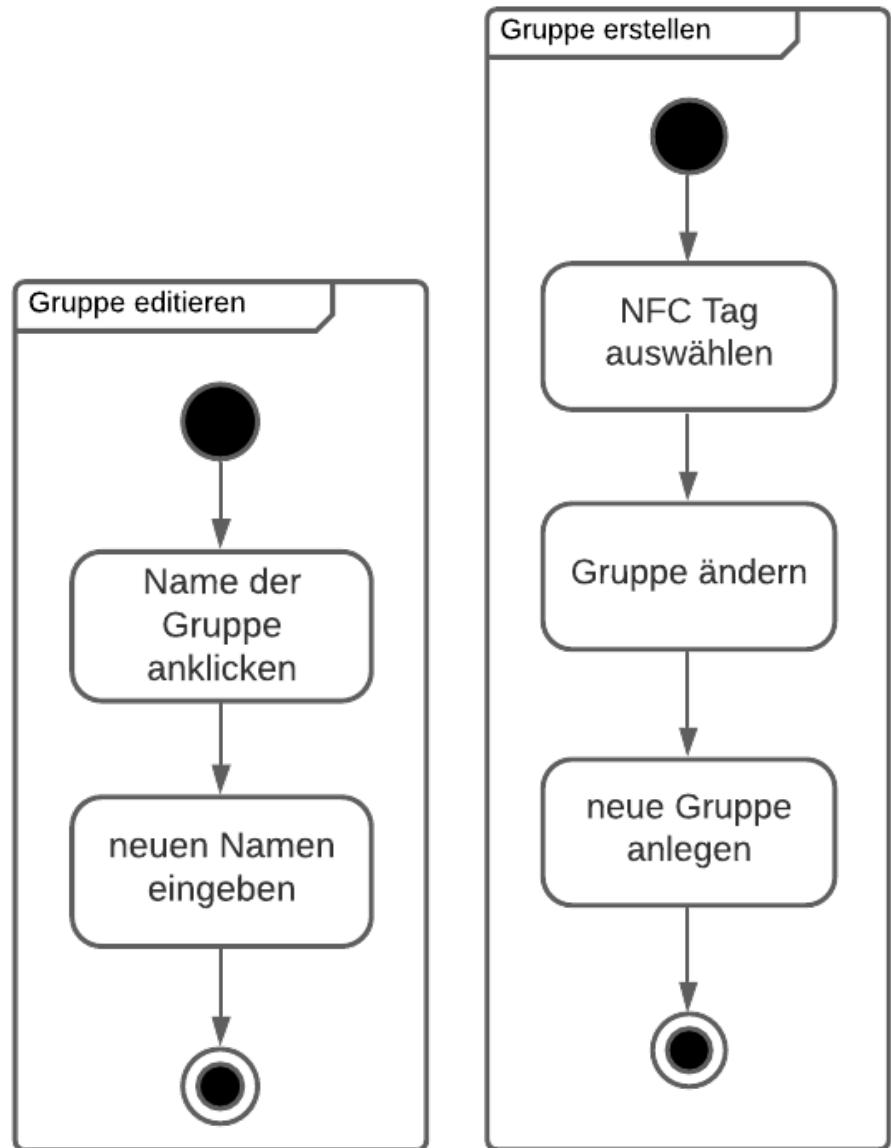
Ich als Benutzer möchte

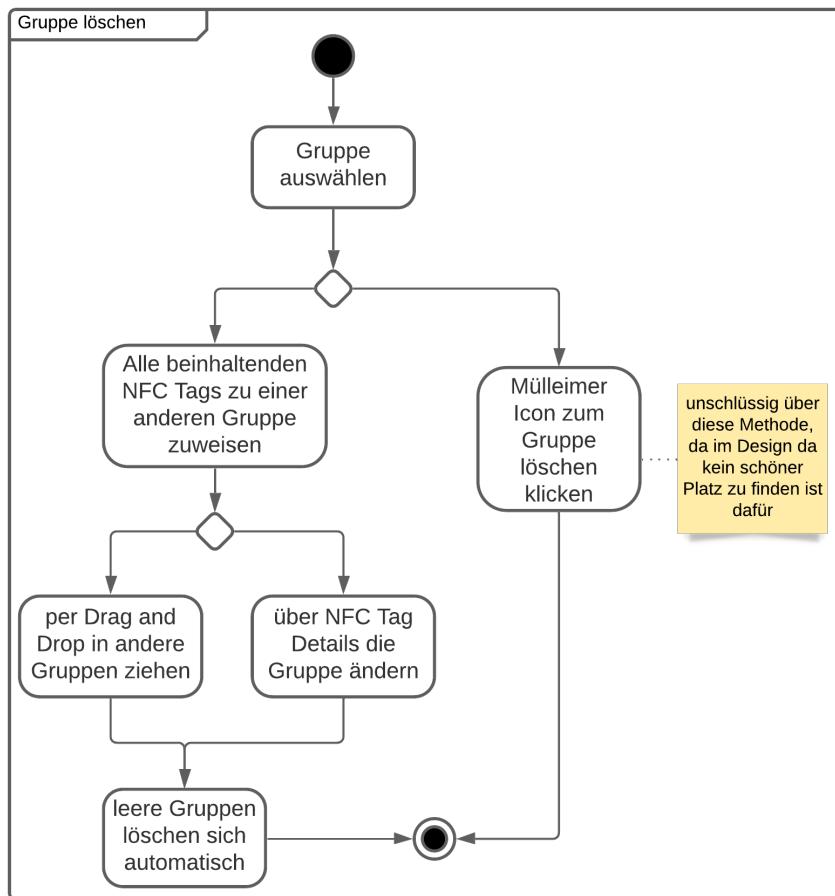
- meine Box mit dem Wlan verbinden
- mich möglichst bequem meine Box verwalten, damit es nicht so lange dauert und ich es nebenbei machen kann
  - NFC Tag mit Handy scannen statt URL eingeben
- die Lautstärke der Box ändern können
- den Track wieder stoppen können
- auswählen auf welchem Gerät es abgespielt wird
- den Song von einem NFC-Tag festlegen / wechseln können
- den einen NFC-Tag löschen können
- meine Tags in einer Übersicht / Liste einfach wiederfinden können.
  - Kamera Bild, Bild upload, Bild URL, Name
- die Einstellungen von einem NFC Tag auf einen anderen übernehmen können
- mich mit Spotify und Youtube registrieren und einloggen können
- weitere Music Services zu meinem existierenden Account hinzufügen können
- nach Tags mit deren Name, Tag-id oder durch ranhalten des Tags an den NFC-Reader in der App suchen können
- alle meine Daten löschen können, um meine Box bedenkenlos verkaufen zu können
- eine Home Seite sehen (wenn ich noch nicht eingeloggt bin) die mich mit einer Info wo ich gelandet bin (Welche App, Was soll ich hier?) empfängt und nicht direkt zum einloggen schickt
- lernen / wissen, wie ich die App benutzen kann
- auch als kostenloser Spotify Benutzer die App verwenden können
- in der App aktiv darauf hingewiesen werden, sollte etwas mit der Konfiguration nicht stimmen oder eine Fehler aufgetreten sein (z.B. wenn mein Tag eingelesen wird)

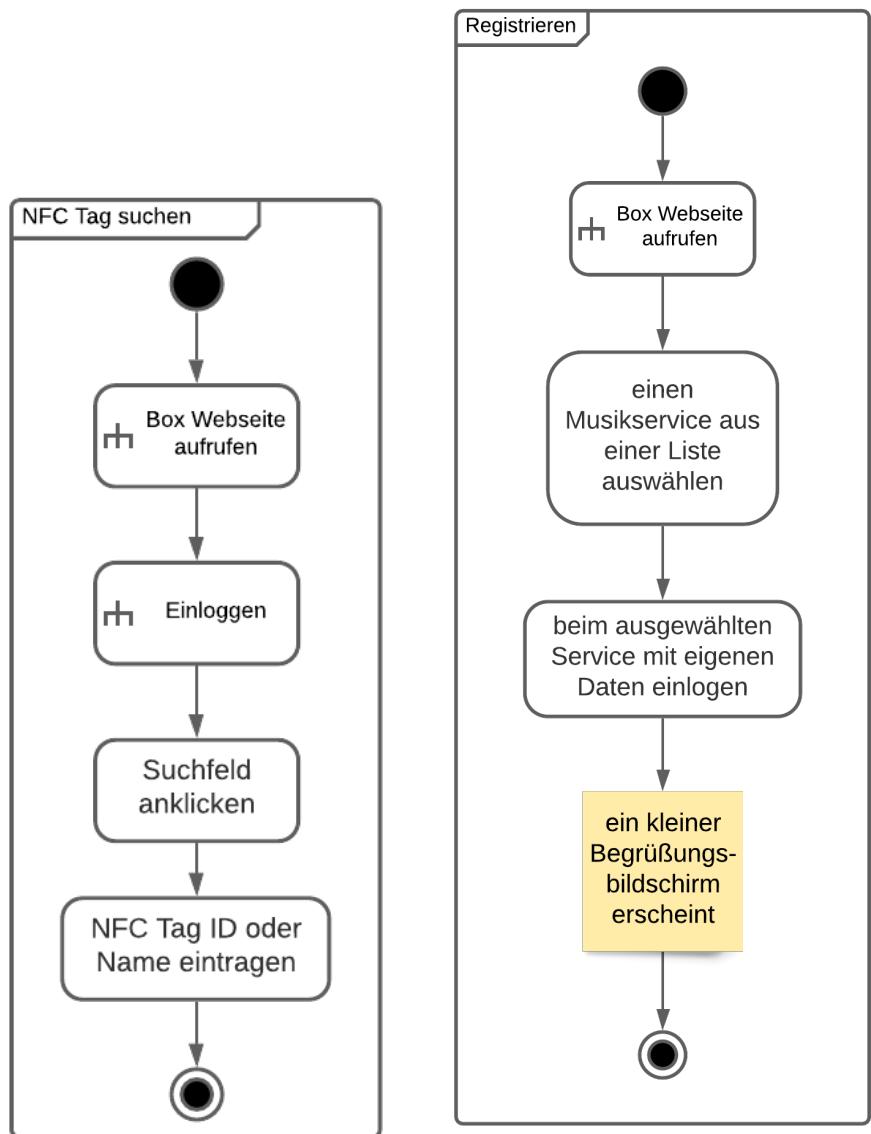
## Use-Flows, die aus den User-Stories entwickelt wurden

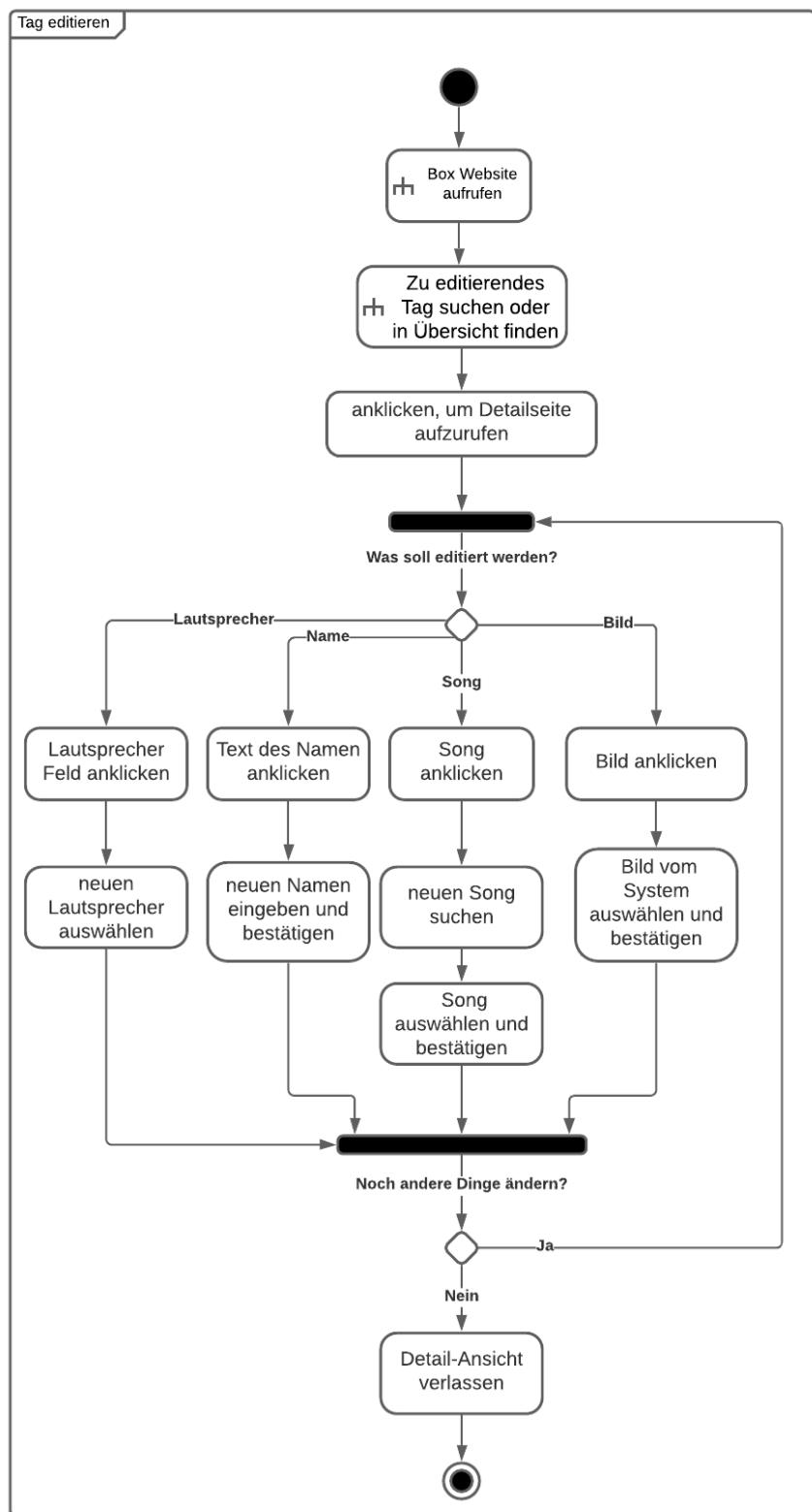


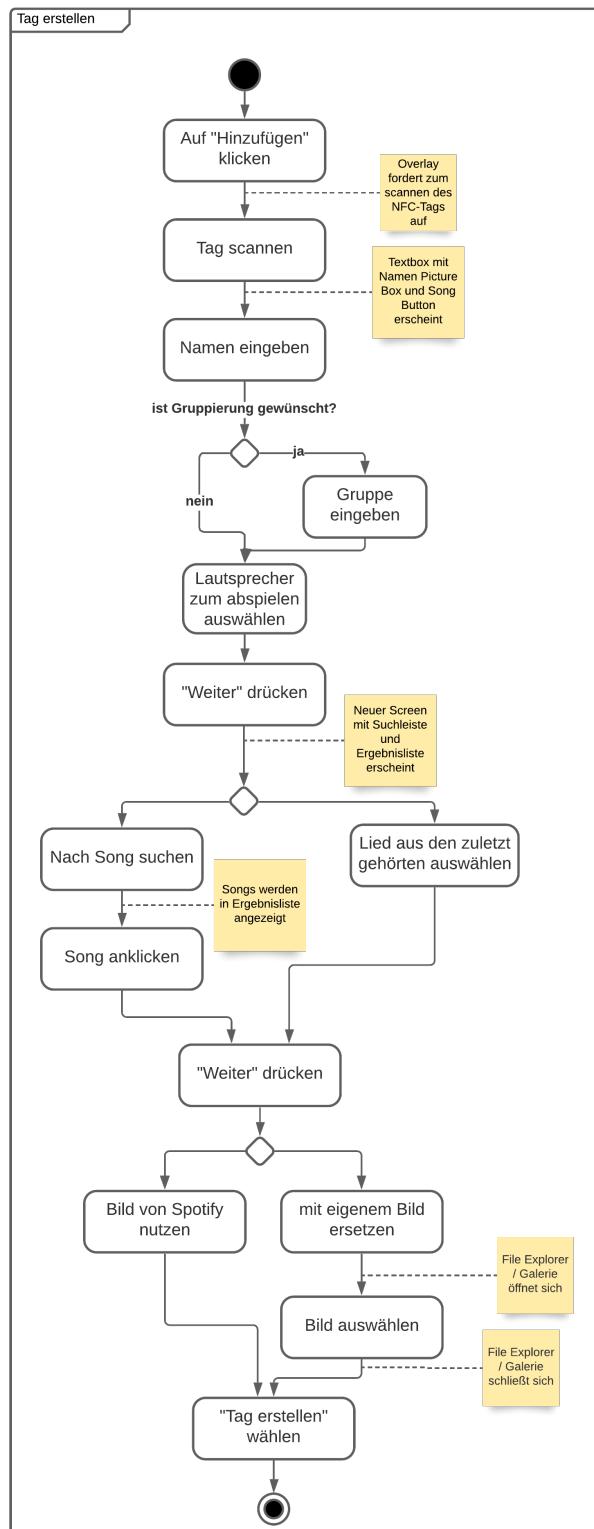


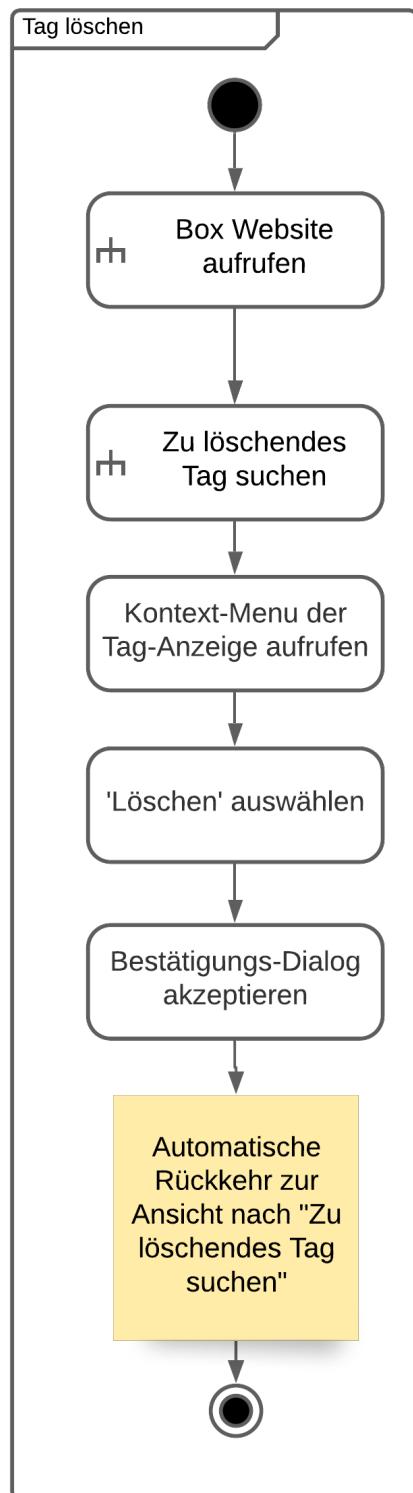






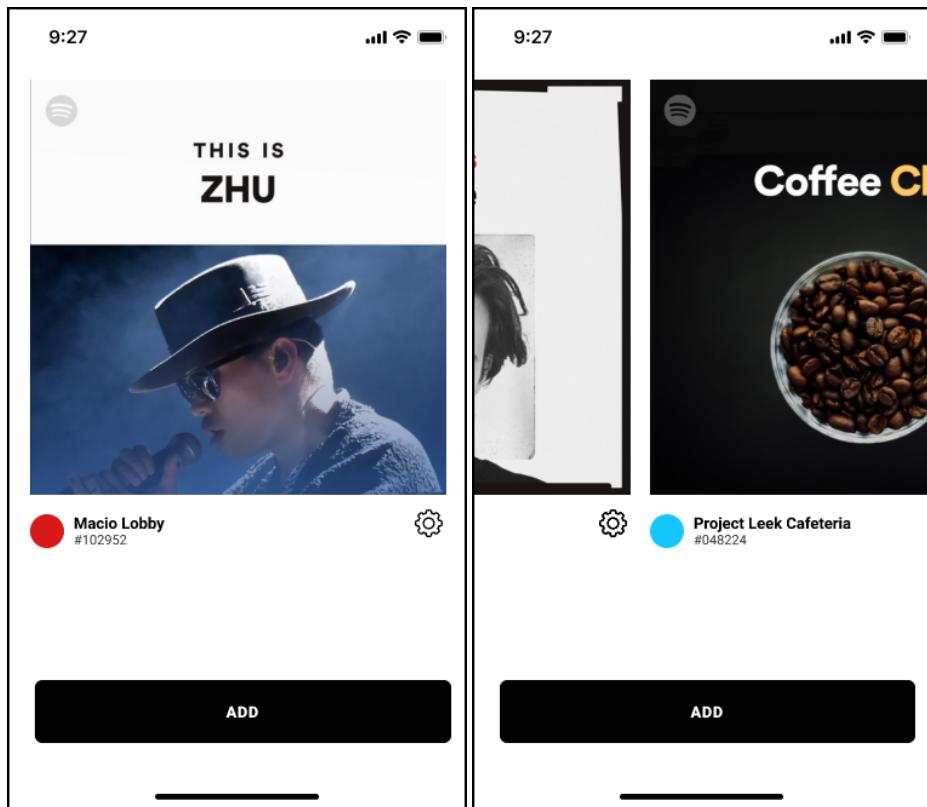






## A.4 Mockups

### Erstes Mockup



## Zweite und dritte Version des Mockups

