Projektbericht Smart Music Player

Anton Bracke
Jan Eberlein
Tom Calvin Haak
Julian Hahn
Nick Loewecke

15. Februar 2021

Inhaltsverzeichnis

1 Einleitung

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

1.1 Unternehmen

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

Das Unternehmen macio stellte in ihrer Ausschreibung als einzige Firma die Anforderung, dass Projekt quelloffen (*Open Source*) umzusetzen. Damit zeigten sie die Bereitschaft, die vom Team geleistete Arbeitszeit nicht für den Eigenbedarf zu nutzen, sondern einen Beitrag für die Open-Source-Community zu leisten. Ihr Hauptgeschäft ist das Design und Entwickeln von Mensch-Maschine-Interfaces. Als etabliertes Unternehmen konnten Sie neben der Rolle des Kunden auch ihre jahrelange Erfahrung bei Bedarf einbringen und Hilfestellung leisten. So gab beispielsweise ein Designer des Unternehmens Feedback zum Entwurf der Benutzeroberfläche

1.2 Projektidee

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

muss auch eine aussagekräftige, öffentliche Dokumentation verfasst werden. Macio stellt die benötigte Hardware zur Verfügung und unterstützt bei technischen Fragen.

1.2.1 Minimal Requirements

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

1.2.2 Stretch Goals

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

2 Projektplanung

2.1 Projekt Management

Am Anfang des Projekt entschied sich das Team in Abstimmung mit macio für eine agile Projektorganisation. Diese Richtung des Projektmanagements zeichnet sich vor allem durch fortlaufende Produktentwicklung, kontinuierliches Feedback, kooperatives Arbeiten und Reaktionsfähigkeit bei Anforderungsänderungen aus. Diese Entscheidung geschah aus mehreren Gründen. Zum einen haben agile Methoden kein festes Endprodukt als Ziel, wie es beispielsweise bei klassischer Softwareentwicklung (Wasserfallmodell, V-Model, etc) der Fall ist. Solch ein festes Ende würde der geforderten Veröffentlichung als Open-Source-Projekt nicht gerecht werden, da solche per Definition erweiterbar sind. Des weiteren haben agile Entwicklungsprozesse vergleichsweise kürzere Veröffentlichungszyklen. Dies ermöglichte dem Team ein funktionierendes Produkt innerhalb des recht knappen Zeitrahmens eines Semesters[1] zu erstellen.
Entsprechend der Entscheidung zu agiler Entwicklung wurde zu Beginn des Projekt keine feste Planung des Projektverlaufs aufgestellt. Stattdessen entwickelte das Team eine Methodik um iterativ im Projekt zu arbeiten. Orientiert wurde sich hierbei an der Methode Scrum. Aus dieser wurden vor allem die Einteilung in Projektiterationen fester Länge (Sprints) und die zugehörigen wiederkehrenden Meetings übernommen. Diese Sprint waren im allgemeinen zwei Wochen[2] lang. Die Rollen der Teammitglieder in Scrum wurde nicht adaptiert, da sich im Team für eine Gleichverteilung der Aufgaben entschieden wurde.

[1] Dieses Semester wahr aufgrund besonderer CoViD19-bedingten Auflagen einige Wochen kürzer als ein typisches.

[2] Abgewichen wurde von dieser Länge nur zum Jahresende, um die Feiertage unterzubringen.

2.1.1 Vor den Sprints

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

2.1.2 Arbeit in den Sprints

Während der Sprints wurde die Produktvision aus den Sprint-Planings implementiert. Alle Mitglieder des Team waren an dieser Arbeit beteiligt. Die einzelnen Arbeitspakete wurden entweder alleine, in Paaren oder selten auch in Gruppen bearbeitet. Dies wurde anhand der Anforderung und des Aufwands der jeweiligen Arbeitspakete entschieden. Auch das spezifische Fachwissen der beteiligten Personen hatte einen Einfluss auf die Aufteilung. In jedem Fall wurden die Pakete erst bei Arbeitsbeginn und nur von den bearbeitenden Mitgliedern selbst zugewiesen.
Nach Bearbeitung eines Paketes wurden die vollzogenen Änderungen durch Code Reviews von mindestens zwei anderen Entwicklern geprüft. Nur wenn diese erfolgreich waren, wurde der entsprechende Code ins Produktinkrement übernommen. Dieses Verfahren wurde angesetzt, um sowohl Flexibilität von Arbeitszeiten zu ermöglichen, als auch um die gewünschte Produktqualität sichherzustellen. Verschiedene Arbeitszeiten waren notwendig, da alle Teammitglieder verschiedene parallele Veranstaltungen besuchten und anderen beruflichen Tätigkeiten nachgingen. So waren keine langfristigen synchronen Arbeitszeiten aller Mitglieder möglich. Die freie Verteilung von Paketen ermöglichte auch, dass Mitglieder in persönlich bevorzugten Themengebieten arbeiten konnten. Dies half die Motivation des Teams am Projekt hochzuhalten.
Um sich während der Sprints abzustimmen und um Arbeitsfortschritte abzugleichen, trafen sich die Teammitglieder drei Mal pro Woche. Dies geschah immer montags, mittwochs und freitags. Bei Bedarf wurde auch von dieser Taktung abgewichen. In Aufbau und Zweck orientierten sich diese regelmäßigen Meetings an den „Daily Standups“ der Scrum-Methode. Während den Meetings stellte jedes Teammitglied kurz vor was es seit dem letzten Standup am Projekt bearbeitet hatte, welche Probleme dabei aufgetreten waren und was bei der Arbeit gelernt wurde. Auch welche Themen jedes Mitglied bis zum nächsten Standup bearbeiten wollte, wurde besprochen. Nach diesen Kurzvorstellungen wurden einzelne besonders interessante und wichtige Punkte der Arbeit oder gelöste Probleme im Detail besprochen. So konnte das Team auf den gleichen Wissensstand kommen und gemeinsam Entscheidungen treffen.

2.1.3 Am Ende der Sprints

Nach jedem Sprint wurde das Ergebnis bzw. das Produktinkrement dem Kunden und der Projektbetreuung vorgestellt. Hier wurde meist auch zusammen mit macio das Ausmaß des nächsten Produktinkrements besprochen. Die resultierenden Wünsche und Rückmeldungen wurden mit ins Backlog und die

folgenden Besprechungen mitgenommen. Im Anschluss fand immer eine Retrospektive statt. In dieser besprach das Team die eigene Zusammenarbeit und den gemeinsamen Umgang. Hierfür stellten sich alle Teammitglieder für sich folgende Fragen für die Arbeit am entsprechenden Sprint:

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

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

2.1.4 Open-Source

Teil der Anforderung war, dass die Software als Open-Source-Projekt der Öffentlichkeit zur Verfügung stehen soll. Dementsprechend entwickelte das Team den Anspruch, das dieses Projekt nicht nur Open-Source sondern auch erweiterbar und verständlich sein soll. Aufgrunddessen wurde die Entscheidung getroffen, das Projekt von Entwicklungsbeginn öffentlich auf GitHub aufzusetzten. Diese Platform bot sich an, da sie für den Umfang des Projekts viele hilfreiche Funktionalitäten in den Bereichen Automatisierung und Kollaboration bietet, allerdings trotzdem kostenlos ist Diese Werkzeuge halfen dabei den Entwicklungsprozess sehr intuitiv und zentral zu gestalten. So wurden zum Beispiel die meisten Diskussionen zu fachlichen Themen direkt in den Arbeitspaketen. Auch im Nachhinein können dann Entscheidungen und Denkprozesse von anderen Kollaborierenden nachvollzogen werden. Dies hilft zusätzlich bei der zukünftigen Instandhaltung und Weiterentwicklung des Projekts.
Das Produkt-Backlog wurde auch in dieses GitHub Repository eingepflegt. Dies erhöhte einerseits die Übersichtlichkeit über den Stand des Projekts innerhalb, da Arbeitspakete und zugehöriger Code sehr einfach verknüpft werden konnten. Andererseits ermöglicht diese Zusammenlegung auch, dass andere Entwickler:innen einfach ins Projekt einsteigen und ihre Arbeit dokumentieren können, ohne andere Plattformen aufsuchen zu müssen. Die Möglichkeiten der Automatisierung wurden genutzt, um den Workflow zu vereinfachen und um die Code-Qualität sicherzustellen. Dies geschah zum Beispiel die durch die automatische Erstellung von Branches für einzelne Arbeitspakete bei Beginn der Bearbeitung und automatische statische und dynamische Tests des Codes.
Darüber hinaus war die Open-Source Anforderung auch hilfreich bei der Produktentwicklung. So konnte die Zielgruppe für dieses Projekt auf Menschen mit fortgeschrittenen Technikkenntnissen und Interesse an Bastelprojekten eingeschränkt werden. Dies war vor allem für das erstmalige Einrichten und die zugehörige Dokumentation maßgebend. Entsprechend konnte sich hier eher technischer Sprache bedient werden. Allerdings galt diese Einschränkung der Zielgruppe auch nicht für alle Breiche des Projekts. Die Benutzbarkeit der Nutzeroberfläche sollte einfach und ohne Vorkenntnisse möglich sein, da die Musikbox z. B. verschenkt oder in Familien genutzt werden könnte. Dieser Anwendungsfall schließt andere Nutzende ein, als bei der Einrichtung.

3 Machbarkeitsstudie

3.1 NFC Tag

3.1.1 lesen

Um mit NFC Tags arbeiten zu können, müssen diese auch entschlüsselt bzw. gelesen werden können. Hierfür ist ein Hardware NFC-Reader notwendig, der die Daten ausliest und an die Box kommuniziert. Dieser emuliert dafür Keyboard Eingaben, die die Tag-IDs darstellen. *Evdev*, ein Kernel Modul von Linux, könnte dann zum Abgreifen dieser Keyboard-Eingaben genutzt werden. Mit *node-evdev*[3] kann *evdev* auch mit Node genutzt werden. Mit dem Fork[4] von Anbraten wird zusätzlich der Raspberry Pi und die Typescript Unterstützung zur Verfügung gestellt.

3.1.2 schreiben

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

3.2 Raspberri Pi

3.2.1 Docker Integration

Auf einem Raspberri Pi Docker zu installieren und zum Laufen zu bringen wird in vielen Anleitungen online beschrieben[5].

3.2.2 Öffentlich zugängliches Web Interface

Auf einem Raspberri Pi könnte eine Web Anwendung gehosted werden, welche die allgemeine Web Anwendung für alle Boxen darstellen soll. Um diese Web Anwendung von außerhalb des eigenen Netzwerkes erreichen zu können, muss innerhalb des Routers ein Port ge-forwarded werden. Dann kann der Pi und dessen Webinterface unter der öffentlichen IP xxx.xxx.xxx.xxx des Routers erreicht werden. Da sich die öffentliche IP-Adresse eines privaten Internet-Anschlusses in der Regel täglich ändert, wird zum einfachen finden der IP ein DynDns Service benötig, welcher eine feste Domain in die wechselnde IP Adresse des Routers übersetzt. Alternativ ginge es auch ohne Port-Forwarding mit nginx und ngrok [6]. Für Unerfahrene wären diese notwendingen Schritte zu Beginn etwas komplexere Thematiken. Der effektive Arbeitsaufwand hängt daher auch sehr stark von der Erfahrung der einzelnen Teammitglieder ab.

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

3.2.3 URL für UI festlegen

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

3.3 User Interface

3.3.1 Zugriff auf NFC Reader von Cloud Anwendung

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

3.3.2 Login via Spotify, Youtube, etc.

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

3.3.3 Gleichen Nutzer bei verschiedenen Loginvarianten wiedererkennen

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

3.3.4 Musik Artwork laden

Sollte bei der Verwendung von Spotify kein Problem sein, da zu jeder Anfrage von Titeln oder Liedern auch eine Liste von Bildern enthalten ist.[7]

3.3.5 Eigene Bilder hochladen

Eigene Bilder hochzuladen sollte möglich sein. In unserem Kontext mit Vue.js und Node.js würde das Plugin *vue-picture-input* helfen. Mit einem Axios Post könnte das Bild an das Backend gesendet werden. [8]

3.3.6 Spotify Connect Lautsprecher auswählen

Das Auswählen von einem spezifischen Spotify Connect Lautsprecher ist möglich. Über einen API Call an die Spotify API mit dem Endpunkt */v1/me/player*

[7] https://stackoverflow.com/questions/10123804/retrieve-cover-artwork-using-spotify-api
[8] https://www.digitalocean.com/community/tutorials/vuejs-uploading-vue-picture-input

/device wird eine Liste von allen verbundenen Geräten geliefert. Über den Endpunkt kann ein entsprechendes Lied zum Abspielen über den jeweiligen *Spotify Connected Speaker* übergeben werden. Sollte nicht explizit ein Lautsprecher angegeben werden, so wird der zuletzt aktive genutzt. Dieser hat bei *is_ active* den Wert *true*. [9]

3.3.7 Spotify Connect Lautsprecher speichern

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

3.3.8 In der Cloud Anwendung die eigene Box auswählen / verbinden

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

3.3.9 Boxdaten über Cloud Anwendung ändern

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

3.3.10 Unterstützung von Youtube Music

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

3.3.11 Unterstützung von Youtube

Youtube bietet die Möglichkeit, nach Videos zu suchen [10]. Um diese Videos auf dem Raspberry als Musik abzuspielen würde sich *youtube-dl* zum downloaden der Videos als *.mp3* Dateien und *omxplayer* zum Abspielen anbieten. Hierfür wäre allerdings ein entsprechender Lautsprecher am Raspberry erforderlich. Ein vergleichbares System zu Spotify Connect existiert derzeit noch nicht.

3.3.12 Unterstützung von Apple Music

Apple Music bietet hier mit deren MusicKit JS[11] eine Möglichkeit, um Musik abzuspielen.

[9] https://developer.spotify.com/documentation/web-api/guides/using-connect-web-api/
[10] https://developers.google.com/youtube/v3/
[11] https://developer.apple.com/documentation/musickitjs/

3.3.13 Unterstützung von Deezer

Deezer lässt sich vergleichbar zu Spotify über eine API steuern.[12]

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

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

3.4 Sonstiges

3.4.1 3D Print version

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

3.4.2 Sound Wiedergabe auf der Box selbst

Manche Pi Modelle verfügen über einen On-Board Audio Anschluss. Die Wiedergabe über diesen ist qualitativ für Musik meist ungeeignet und sollte daher über ein weiteres Audiomodul oder eine externe Soundkarte erfolgen. Innerhalb der Raspberri Pi Reihe gibt es dafür Accessoires, die circa 20-30€ kosten.[13]. Zur Wiedergabe auf der Box selbst müsste dafür auf dem Raspberry eine Spotify Instanz laufen, damit auch diese als Connected Speaker erkannt wird. Hierfür existieren Libraries wie *raspotify*[14].

3.4.3 Box unter 30€ Kosten

Mit einem Raspberri Pi wäre dieses Ziel möglich, es könnte aber kein Pi ab Model 3 verwendet werden, da diese über dem Ziel liegen. Mit dem Raspberri Pi Zero W mit eingebautem W-Lan und einem USB Port für den NFC-Reader gäbe es ein kostengünstiges Model, welches für ca. 10$ erhältlich ist [15].

4 Design Mockups

TODO: setze pdfs ein

5 Durchführung

5.1 Projektstrukur

[12] http://developers.deezer.com/login?redirect=/api
[13] https://www.raspberrypi.org/products/
[14] https://github.com/dtcooper/raspotify
[15] https://www.raspberrypi.org/products/raspberry-pi-zero-w/

Die an das Projekt gesetzten Anforderungen machten die Erstellung mehrerer Applikationen notwendig. Neben einer Benutzeroberfläche, auf der die NFC-Tags verwaltet werden können, musste die Steuerung des NFC-Readers und ein System zur Speicherung der Daten sowie zum Abspielen der Musik entwickelt werden. Die dafür benötigten Anwendungen wurden als Microservices konzipiert und umgesetzt. Zur Verringerung des Aufwands der Konfiguration und Wartung wurden die einzelnen Microservices in einem Mono-Repository auf GitHub zusammengefasst. Jeder Service wurde in einem Unterdner in *packages* angelegt (vgl. Abbildung 1). Der Ordner *app* enthält die Benutzeroberfläche, *backed* die Datenbankschnittstelle sowie Logik zur Kommunikation mit dem Musik-Streaming-Anbieter und *nfc-reader* die Applikation zum Steuern und auslesen des am Raspberry Pi angeschlossenen NFC-Readers. Im Ordner commons befinden sich von allen Services gemeinsam genutzte Ressourcen (wie z. B. Modell-Klassen). Neben den einzelnen Microservices wurden ebenfalls die Dokumentationsdateien wie Setup-Guide, Benutzerdokumentation und Projektbericht im Mono-Repo im Ordner *docs* abgelegt. Auch die für den Dienst *Docker* benötigten Dateien fanden in dem gleichnamigen Ordner Platz. Die Konfigurationsdateien für *ESLint* und den *Typescript*-Compiler sind im Ordner *config* zu finden. Auf die Auflistung aller weiteren Dateien wird in diesem Bericht verzichtet. Eine genaue Aufstellung ist dem *GitHub-Repository*[16] zu entnehmen. Durch die Zusammenfassung in ein Mono-Repository waren die einzelnen Services für jeden Entwickler jederzeit einfach erreichbar, ohne die Notwendigkeit das Repository zu wechseln.

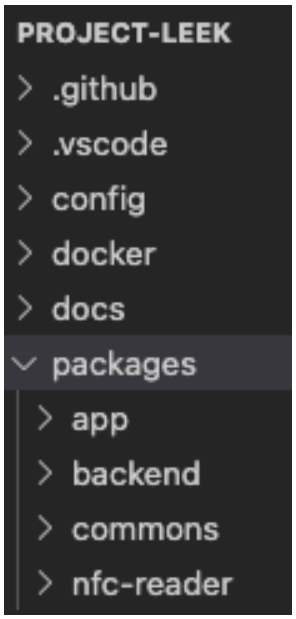

Abbildung 1: Orderstruktur

5.2 Technologien

5.2.1 Programmiersprachen

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

[16] `https://github.com/project-leek/project-leek`

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

5.2.2 Frameworks

Neben der Programmiersprache Typescript setzte das Team verschiedene Frameworks ein, um bei der Entwicklung auf bereits existierenden Lösungen mit weniger Aufwand zu entwickeln.

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*[17] Dieses wurde aufgrund seiner guten Dokumentation und seiner flachen Lernkurve gewählt, wodurch sich der Einstieg für Teammitglieder mit weniger Erfahrung in der Web-Entwicklung leichter gestaltete. Außerdem bietet das in Vue.js verwendete Entwurfsmuster MVVM (Model View ViewModel) den Vorteil, dass die designaffineren Entwickler sich eher mit dem Frontend beschäftigen können, während andere am Logik-Code arbeiten. Im Vergleich zu anderen Webframeworks gilt es, laut Benchmarks außerdem als sehr performant.[Car20]

Tailwind CSS
Zur effizienteren Gestaltung der Benutzeroberfläche entschied sich das Team für das Utility-First-Framework *TailwindCSS*[18]. 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. [Sto]

FeathersJS
Bei der Erstellung des Backends entschied sich das Team zusätzlich das Framework *FeathersJS*[19] einzusetzten. Dies ermöglicht komfortable CRUD-Zugriffe[20] 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. Abb. 2) Dabei können verschiedenste Protokolle, wie HTTP oder WebSockets 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 auch Server jederzeit mit der Datenübertragung beginnen, ohne vorher - wie bei HTTP - jedes Mal eine neue Verbindung aufzubauen zu müssen. [Mal] Dadurch kann der Server alle verbundenen Clients bei einem CRUD-Zugriff informieren,

[17] `https://vuejs.org/`
[18] `https://tailwindcss.com`
[19] `https://feathersjs.com/`
[20] CRUD = **C**reate-**R**ead-**U**pdate-**D**elete

```
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 2: Implementierung eines ServicesTest

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

5.2.3 Containervirtualisierung (Docker)

Es wurde lediglich die Anforderung gestellt, dass die Applikation vor allem für mobile Endgeräte optimiert sein soll. Auf Basis dieser Anforderung evaluierte das Projektteam bekannte Technologien und recherchierte mögliche Alternativen. Um die Installation der Box möglichst komfortabel zu gestalten, wird auf die freie Containervirtualisierungssoftware *Docker*[21] gesetzt. Ohne diese Möglichkeit wäre die Installation aufgrund der drei Microservices (Backend, Reverse-Tunnel und NFC-Reader) sehr aufwändig. Der Vorteil von Docker besteht darin, dass das Team lediglich eine sogenannte *docker-compose* Datei benötigt, in dem die Konfiguration der Docker-Container beschrieben ist. Es enthält einen Verweis auf die jeweiligen Container-Images, auf denen die Container aufbauen (z. B. leek-backend). Diese Images enthalten bereits alle notwendigen Abhängigkeiten und Programme, sodass weitere Installationen seitens der Benutzer:innen nicht erforderlich sind. Außerdem lässt sich das Verhalten der Images durch Umgebungsvariablen weiter konfigurieren. Durch die einfache Auslieferung bietet Docker außerdem den Vorteil der Reproduzierbarkeit. So können aufgetretene Fehler problemlos von einem Entwickler nachgestellt werden, da Docker das Betriebssystem des Anwenders von der benötigen Umgebung der leek-box abstrahiert.

[21] https://www.docker.com/

5.2.4 Reverse-Tunnel (ngrok)

Um verschlüsselt vom Frontend (unter `https://project-leek.github.io` erreichbar) auf das jeweilige Backend einer *leek-box* zuzugreifen, wird der Reverse-Tunnel Dienst *ngrok*[22] verwendet. Dies ist notwendig, da moderne Browser eine verschlüsselte Verbindung (mit SSL-Zertifikat) zu allen Komponenten auf der Website verlangen, sobald die Website selbst per SSL geladen wird. Da das Framwork jedoch nativ kein HTTPS unterstützt, wird *ngrok* verwendet, um die unverschlüsselten Daten durch einen verschlüsselten Tunnel zum Frontend zu schicken. Damit wird die Anforderung der Verschlüsselten Verbindung erfüllt. Außerdem soll das Backend unabhängig vom lokalen Netzwerk erreichbar sein, ohne dass eine umständliche Portweiterleitung eingerichtet werden muss. Um dies zu leisten, baut das Backend einen Tunnel zu dem Server mit der bekannten Adresse `ngrok.io` auf. Dabei wird eine zufällige Subdomain im Schema *xyz.ngrok.io* angelegt und dem Benutzer in der Konsole des Backends angezeigt. Diese Adresse kann der Benutzer nun im Frontend (`https://project-leek.github.io`) eingeben, um sich mit seiner Box zu verbinden. Hier Schaubild?

5.2.5 Externe Bibliotheken

Spotify Web API
Um die abspielbaren Songs inklusive Cover-Bildern zu ermitteln und diese abzuspielen ist ein Zugriff auf die API eines Musikstreaming-Dienstes notwendig. Der vom Kunden vorgeschlagene Anbieter war *Spotify*[23]. Um den Zugriff auf die Spotify API zu simplifizieren wurde die Bibliothek *Spotify Web API Node*[24] verwendet. Diese bietet bereits die benötigten Methoden, wie z. B. die zum Suchen eines Songs anhand eines Suchbegriffs. Neben dem Songtitel und der Spotify-Url werden auch die Künstler und die Adresse des Albumcovers mitgeliefert und konnten von den Entwicklern ohne Mehraufwand genutzt werden. Auch eine Methode zur Ermittlung der verfügbaren Geräte zum Abspielen der Musik stellt die Bibliothek bereit.

NeDB
Als Datenbank für dieses Projekt wurde die kostenlos verfügbare JavaScript-Datenbank *NeDB*[25] gewählt. Sie wird verwendet, um die NFC-Tags, Benutzer und den angeschlossenen NFC-Reader zu verwalten. *NeDB* ist eine auf *MongoDB*[26] aufbauende, sehr schnelle JSON Datenbank. Sie wurde gewählt, da sie eine geringe Komplexität besitzt und weil *FeathersJS* einen Datenbankadapter für diese Datenbank bereitstellt, was den Zugriff auf die Datenbank sowie die Verwaltung der Daten erleichterte. Außerdem arbeitet die Datenbank inkrementell, was das Risiko auf sehr große Datenbankdateien minimiert. Diese Eigenschaft war hilfreich, um die Datenbank auf einem Raspberry Pi betreiben zu können. Zu Beginn des Projekts diskutierte das Team die Technologien, mit denen die *leek-box* umgesetzt werden sollte. Der Kunde ließ dem Projektteam hierbei sehr viele Freiheiten.

[22] `https://ngrok.com/`
[23] `https://www.spotify.com`
[24] `https://github.com/thelinmichael/spotify-web-api-node`
[25] `https://github.com/louischatriot/nedb`
[26] `https://www.mongodb.com/`

5.3 Hilfsmittel

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

5.3.1 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 dieses Projekt die ideale Umgebung zur Quellcodeverwaltung und Kollaboration dar.

Kollaboration
Mit dem Anspruch das Produkt auch nach Projektende fortzuführen, wurden auch die Projektmanagement Funktionen von GitHub verwendet. Die einzelnen Aufgaben bzw. Arbeitspakete (*Issues*[27]) wurden in einem *Issue Board* den verschiedenen Arbeitsstatus[28] zugeordnet. Außerdem konnten verschiedene Informationen, wie z. B. der Sprint bzw. *Milestone*[29] in dem das Ticket abgeschlossen sein soll oder der bearbeitende Entwickler dokumentiert werden. Da diese Ansicht ebenfalls nach diversen Eigenschaften gefiltert werden kann, konnte sich jeder Entwickler schnell einen Überblick über den aktuellen Projektstand verschaffen. Das Team setzte zur Unterstützung außerdem den *Renovate-Bot*[30] ein, welcher die Tickets automatisch anhand von verschiedenseten Events verschob und einen Branch anlegt, wenn sich ein Entwickler einem Ticket zuordnet.

Eine weitere Funktionalität von GitHub, die in diesem Projekt genutzt wurden sind die sogenannten *Actions*[31]. Sie ermöglichen automatisierte Tests, die den Quellcode und die Anwendung als Gesamtkonstrukt auf verschiedenste Fehler prüfen und frei konfigurierbar sind. In diesem Projekt wurden die folgenden vier Tests durchgeführt um die Änderungen vor der Übertragung in den Master-Branch zu validieren:

1. **build**: Wird die Software fehlerfrei gebaut?
2. **lint**[32]: Ist der Quellcode in gutem Stil geschrieben?
3. **test**: Sind die geschrieben Unit-Tests erfolgreich?
4. **typecheck**: Sind alle Typen kompatibel?

[27] Da das Team größtensteils den Begriff *Issue* nutzte, wird dieser auch fortlaufend hauptsächlich verwendet.
[28] Status = Backlog, Todo, In Progess, Review und Done
[29] Auch hier wird im weiteren Verlauf eher der Begriff Milestone oder Meilenstein verwendet.
[30] `https://github.com/renovatebot/renovate`
[31] `https://github.com/features/actions`
[32] Tool zur statischen Codeanalyse

Sind alle Tests erfolgreich, ist die Änderung technisch nutzbar.

Außerdem wurde die in GitHub existierende Funktionalität der Pull Requests verwendet. Beim *Pushen*[33] werden die Änderungen automatisch in dem den Branch betreffenden Pull Request angezeigt (wenn vorhanden). Jeder Pull Request enthält allgemeine Informationen, welchem Zweck er dient und von wem die Änderung durchgeführt wird. Außerdem wird das Ergebnis der automatisierten Tests (*GitHub Actions*) angezeigt. Das Projekt ist so konfiguriert, dass die Übernahme der Änderung des Pull Requests nur möglich ist, wenn mindestens zwei andere Entwickler diesen Änderungen durch Code Reviews zugestimmt haben und alle automatisierten Test erfolgreich abgeschlossen wurden. Ist dies der Fall, kann die Änderung in den Master-Stand übertragen werden. Für die Code-Reviews wurde ebenfalls die GitHub interne Funktionalität genutzt, die bequeme Änderungsvorschläge ermöglicht.

Github Pages
Zum Bereitstellen des Frontends der *leek-boxen* wird der Dienst GitHub Pages verwendet. Dieser stellt pro Organisation/Account eine kostenlose Website zur Verfügung. Die Konigurationsanforderungen für diese Website sind minimal, da der Inhalt der Website an einen Teil des Repositotys gebunden werden kann. In unserem Fall, wird hierfür das *app-package* verwendet. So bleibt die Website immer aktuell, da sie auf einem eigenen Branch besteht, und der Stand der Website komfortabel durch einen merge vom Master-Branch in den Website-Branch aktualisiert werden kann. @Anton Please check!

5.3.2 Visual Studio Code & WSL

Als Entwicklungsumgebung in diesem Projekt wurde *Visual Studio Code* verwendet. VSCode war allen Gruppenmitgliedern bereits bekannt und wurde von vielen auch regelmäßig verwendet. Die IDE beherrscht den Umgang mit allen für das Projekt benötigten Dateitypen und ist durch eine Vielzahl an kostenlos angebotenen Erweiterungen (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 *Visual Studio Code* eine komfortable Anbindung[34] an das *Windows Subsystem für Linux (WSL)*[35], welches eingesetzt wurde, um eine möglichst homogene Arbeitsumgebung innerhalb des Teams sicherzustellen und neben der gleichen IDE auch auf Betriebssystemen der gleichen Kernel-Infrastrukur (UNIX) zu arbeiten. Dies erleichterte insbesondere die Arbeit mit *Docker* deutlich.

5.3.3 Prototyping (Figma)

Zur Abstimmung auf ein Design für die Benutzeroberfläche wurden vor der Entwicklung Prototypen der benötigeten Komponenten mittels des

[33] Hochladen der Änderung vom Entwickler-PC auf das Repositoty auf GitHub
[34] `https://code.visualstudio.com/docs/remote/wsl`
[35] `https://docs.microsoft.com/de-de/windows/wsl/about`

webbaiserten Prototyping-Tools *figma*[36] konzipiert. So konnten neben dem Design auch Abläufe in Form von „Click-Dummys“ erstellt werden und mit dem Kunden abgestimmt werden. *Figma* wurde verwendet, da es einen kollaborativen Zugriff ermöglicht, kostenlos ist und bereits Erfahrung mit der Plattform bestand

5.3.4 Lucidchart

Um Abläufe schon vor dem Prototyping skizziert darstellen zu können wurde außerdem die webbasierte Plattform *Lucidchart*[37] verwendet. Hier wurde neben der initial erstellten Produktübersicht auch mehrere Aktivitätsdiagramme erstellt, die aus vom Projektteam entwickelten und mit dem Kunden abgestimmten User-Stories entstanden sind. Diese sind im Anhang unter Verweis einfügen! zu finden. Auch diese Plattform ermöglicht den gleichzeitigen Zugriff und war den Entwicklern bereits bekannt, was die Notwendigkeit der Einarbeitung eliminierte.

5.4 Entwicklungszyklus

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

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

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

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

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

[36] https://www.figma.com/
[37] `https://www.lucidchart.com/`

6 Projekt Meilensteine

6.1 Meilenstein 1

TODO: Kurze Einleitung, von wann bis wann ging

6.1.1 Ziel

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

6.1.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.1.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.1.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.1.5 Retrospektive

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

6.2 Meilenstein 3

TODO: Kurze Einleitung, von wann bis wann ging

6.2.1 Ziel

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

6.2.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.2.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.2.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.2.5 Retrospektive

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

6.3 Meilenstein 4

TODO: Kurze Einleitung, von, bis

6.3.1 Ziel

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

6.3.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.3.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.3.4 Erkenntnisse

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

6.3.5 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.3.6 Retrospektive

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

7 Code Walkthrough

TODO: vielleicht interessante Code Snippets?

8 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.

8.1 Statische Tests

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

8.1.1 Linting

Beim Linting wurde der Code auf syntaktische Korrektheit geprüft. Auch manche semantische (laufzeitunabhängige) Aspekte wurden untersucht. So konnten kleinere Denkfehler und Flüchtigkeitsfehler, wie z. B. fehlende Kommata, schnell gefunden und behoben werden. Hierfür wurde das Analyse-Werkzeug *ESLint*[38] genutzt. Für diese Werkzeug war zusätzlich auch eine Erweiterung[39] 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*[40] genutzt. Diese stellte die selben Funktionalitäten für den Vue-Framework spezifischen Code zur Verfügung.
Darüber hinaus boten diese Erweiterungen auch Formatierungshilfen mit übertragbaren Konfigurationen. Durch deren Nutzung konnte ein konsistenter Code-Stil innerhalb des Teams ermöglicht werden. Dies legte den Grundstein für lesbaren Code und ermöglichte successive effizientes Arbeiten.

8.1.2 Reviews

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

8.2 Dynamische Tests

Als dynamische Tests wurden in diesem Projekt Anwendungsfall-basierte Tests genutzt. Bei dieser Art von Test werden auf User-Stories aufbauende Testfälle herangezogen. Diese Testfälle werden Schritt für Schritt „durchgespielt“. Dabei wird überprüft, ob das tatsächliche Ergebnis dem erwartetem entspricht. Die meisten Test dieser Art wurden als Integrations-Tests durchgeführt. Bei diesen wurde die Änderungen im Kontext der bestehenden Software getestet. 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. Diese Tests waren Teil des Deployment Cycles und wurden so in den meisten Fällen parallel zu den Reviews durchgeführt.

Ein gewisser Teil der anwendungsfallbasierten Test fand auf der Ebene von Code-Funktionen (*Units*) statt. Diese Unit-Tests wurden zur einfacheren

[38] `https://eslint.org/`
[39] `https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint`
[40] `https://vuejs.github.io/vetur/`

Wiederholbarkeit automatisiert. Dazu wurde das JavaScript-Test-Framework *Jest*[41] genutzt. Dieses wurde gewählt, da sich Tests hiermit sehr intuitiv und ohne großen Lernaufwand schreiben ließen. Auch diese Tests waren Teil des Deployment Cycles und wurden so bei jedem Pull-Request automatisch ausgeführt.

9 Fazit

9.1 Probleme, Lösungen, Erkenntnisse

TODO: Finde TITEL: Fazit oder nur Erkenntnisse oder oder? Besseres Intro zum Kapitel? Das Projekt Informatik gab einen realistischen Einblick in die Arbeitswelt. Das Team musste sich klassischen Alltagsproblemen stellen, Lösungswege finden und konnte wichtige Erkenntnisse für die Zukunft daraus ziehen. Auch Coding-Entwurfsmuster und -Konventionen wurden neu erlernt und praktisch angewandt.

9.1.1 Einarbeitung in neue Technologien

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

9.1.2 Versionskontrolle und Deployment Cycle

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

[41] `https://jestjs.io/`

9.1.3 Automatisierte Tests und Deployment

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

9.1.4 Regelmäßige Standups und Retrospektiven

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

9.1.5 Kommunikation und Eigeninitiative

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

9.1.6 Planung, Zielsetzung und Einschätzung

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

Branch mergen zu können. Auch wenn die Lösung dieses Problem hätte effizienter behandelt werden können, war diese Methode angesichts der knappen Projektlaufzeit die effektivste.

10 Technische Diagramme

TODO: ER Diagramme, UML, solcher krams

11 Anhang

BEISPIEL, DELETE THIS Buchreferenz [Gus13] oder Seitenref [Gus69]

Literatur

[Car20] Ryan Carniato. *JavaScript Frameworks, Performance Comparison 2020*. 2020. URL: https://medium.com/javascript-in-plain-english/javascript-frameworks-performance-comparison-2020-cd881ac21fce (besucht am 21. 12. 2020).

[Gus13] Hans Gustav. *Dreizehntes Gesetz zur Änderung des Manzkes*. 13. Feb. 2013.

[Gus69] Hans Gustav. *Google*. 2069. URL: http://www.google.de (besucht am 13. 02. 2013).

[Kie21] FH Kiel. *Qualifikationsziele des Studiengangs Informationstechnologien*. 2021. URL: https://www.fh-kiel.de/fachbereiche/informatik-und-elektrotechnik/studiengaenge/bachelor-studiengaenge/informationstechnologie-bachelor/qualifikationsziele/ (besucht am 08. 02. 2021).

[Mal] Eiji Kitamura Malte Ubl. *Das Problem: Client-Server- und Server-Client-Verbindungen mit geringer Latenz*. URL: https://www.html5rocks.com/de/tutorials/websockets/basics/ (besucht am 20. 10. 2010).

[Sto] Kathrin Stoll. *Tailwind – die CSS-Zukunft heißt Utility-First*. URL: https://t3n.de/news/tailwind-css-zukunft-heisst-1299218/ (besucht am 28. 07. 2020).

[Tor] Linus Torval. URL: http://www.linux-kurs.eu/opensource.php (besucht am 14. 02. 2020).

[Ulb] Heinrich Ulbricht. *Drei Gründe warum TypeScript das bessere JavaScript ist*. URL: https://www.communardo.de/blog/drei-gruende-warum-typescript-das-bessere-javascript-ist/ (besucht am 23. 03. 2017).