Projektbreicht 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 „Informationstechnologien“ 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

TODO: was macht macio aus

1.2 Projektidee

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

1.2.1 Minimal Requirements

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

1.2.2 Stretch Goals

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

2 Projektplanung

2.1 Projekt Management

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.

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.

[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.

2.1.2 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.

3.2.3 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] 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/

3.3 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 /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]

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

3.3.7 Spotify Connect Lautsprecher speichern

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

3.3.8 In der Cloud Anwendung die eigene Box auswählen / verbinden

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

3.3.9 Boxdaten über Cloud Anwendung ändern

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

3.3.10 Unterstützung von Youtube Music

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

3.3.11 Unterstützung von Youtube

Youtube bietet die Möglichkeit, nach Videos zu suchen [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.

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.

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

3.4 Sonstiges

3.4.1 3D Print version

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

3.4.2 Sound Wiedergabe auf der Box selbst

Manche Pi Modelle verfügen über einen On-Board Audio Anschluss. Die Wiedergabe über diesen ist qualitativ für Musik meist ungeeignet und sollte daher über ein weiteres Audiomodul oder eine externe Soundkarte erfolgen. Innerhalb der Raspberri Pi Reihe gibt es dafür Accessoires, die circa 20-30€ kosten.[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

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

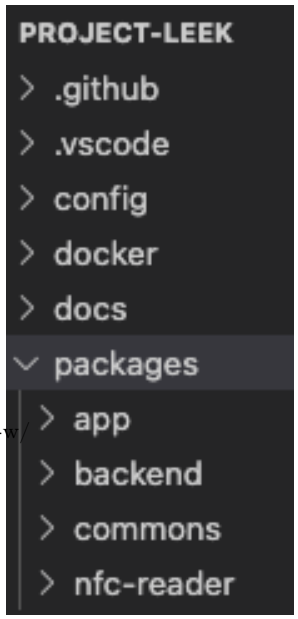

Abbildung 1:
Orderstruktur

[13] https://www.raspberrypi.org/products/
[14] https://github.com/dtcooper/raspotify
[15] https://www.raspberrypi.org/products/raspberry-pi-zero-w/

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.

5.2 Technologien und Hilfsmittel

TODO: Vue, vscode, devops krams, etc Entwickelt wird mit Visual Studio Code, da es eine einfache Nutzung des Linux-Subsystems ermöglicht. [17].

5.3 Deployment Cycle

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

5.4 Probleme während der Durchführung

TODO:

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?

[16] `https://github.com/project-leek/project-leek`
[17] https://code.visualstudio.com/docs/remote/wsl

6.1.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.1.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.1.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.1.5 Retrospektive

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

6.2 Meilenstein 3

TODO: Kurze Einleitung, von wann bis wann ging

6.2.1 Ziel

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

6.2.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.2.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.2.4 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.2.5 Retrospektive

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

6.3 Meilenstein 4

TODO: Kurze Einleitung, von, bis

6.3.1 Ziel

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

6.3.2 Probleme

TODO: welche prrobleme sind aufgetreten?

6.3.3 Lösungen

TODO: Welche Lösungen haben wir gefunden?

6.3.4 Erkenntnisse

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

6.3.5 Product Increment

TODO: Was ist am Ende dabei rumgekommen?

6.3.6 Retrospektive

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

7 Erkenntnisse

8 Code Walkthrough

TODO: vielleicht interessante Code Snippets?

9 Testing

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.

9.1 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.

9.1.1 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*[18]

[18] `https://eslint.org/`

genutzt. Für diese Werkzeug war zusätzlich auch eine Erweiterung[19] 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*[20] 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.

9.1.2 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.

9.2 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 Wiederholbarkeit automatisiert. Dazu wurde das JavaScript-Test-Framework *Jest*[21] 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.

[19] `https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint`
[20] `https://vuejs.github.io/vetur/`
[21] `https://jestjs.io/`

10 Technische Diagramme

TODO: ER Diagramme, UML, solcher krams

11 Anhang

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

Literatur

[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).