University of Applied Sciences

HOCHSCHULE
EMDEN·LEER

Projektdokumentation

Test Driven Development eines Systems zur Erfassung, Bearbeitung und Auswertung von Wettkampfdaten für Schwimmwettkämpfe

Betreuer

Prof. Dr.-Ing. Dirk Rabe
Frederik Gosewehr, M.Eng

Teilnehmer

David Boes
Patrick Brink
Anja Grobelnik
Rouven Hollens
Ina Krefting
Mengyao Lu
René Kuchenbuch
Arne Otten
Henning Pauls
Keno Peil
Marc Pollmann
Tom Wessels

Hochschule Emden/Leer
18.2.2019

Erklärung zu Nutzungsrechten:

Soweit unsere Rechte berührt sind, erklären wir uns einverstanden, dass die vorliegende Arbeit Angehörigen der Hochschule Emden/Leer für Studium / Lehre / Forschung uneingeschränkt zugänglich gemacht werden kann.

Eidesstattliche Erklärung:

Wir, die Unterzeichnenden, erklären hiermit an Eides statt, dass wir die vorliegende Arbeit selbständig verfasst haben und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Alle Quellenangaben und Zitate sind richtig und vollständig wiedergegeben und in den jeweiligen Kapiteln und im Literaturverzeichnis wiedergegeben. Die vorliegende Arbeit wurde nicht in dieser oder einer ähnlichen Form ganz oder in Teilen zur Erlangung eines akademischen Abschlussgrades oder einer anderen Prüfungsleistung eingereicht.

Uns ist bekannt, dass falsche Angaben im Zusammenhang mit dieser Erklärung strafrechtlich verfolgt werden können.

Emden, den __.__.____

_____	_____
David Boes	Patrick Brink
_____	_____
Anja Grobelnik	Rouven Hollens
_____	_____
Ina Krefting	Mengyao Lu
_____	_____
René Kuchenbuch	Arne Otten
_____	_____
Henning Pauls	Keno Peil
_____	_____
Marc Pollmann	Tom Wessels

Inhaltsverzeichnis

1 Einleitung

Die ausgearbeitete Dokumentation bezieht sich auf das Projekt “Test Driven Development eines Systems zur Erfassung, Bearbeitung und Auswertung von Wettkampfdaten für Schwimmwettkämpfe”, welches im Rahmen des Moduls „Projektgruppe“ im Bachelorstudiengang „Informatik“ an der Hochschule Emden/Leer im Wintersemester 2018/2019 erarbeitet wurde.

Die Projektgruppe bestand aus zwölf Studierenden aus dem fünften Semester, die unter der Betreuung von Prof. Dr. Dirk Rabe und M. Eng. Frederik Gosewehr das Projekt selbständig erarbeitet haben.

Die folgenden Kapitel geben Auskunft über den weiteren Projektverlauf und das entwickelte System. Zuerst wird genauer die Aufgabenstellung und das Projektmanagement vorgestellt. Im weiteren Verlauf wird auf das Systemkonzept, die Modellierung und die eigentliche Implementation des Systems sowie der einzelnen Komponenten eingegangen. Anschließend werden das Testkonzept und der Projektverlauf beim ersten Einsatz dargestellt. Im Anschluss werden mögliche Systemerweiterungen, Schwierigkeiten und Empfehlungen für die nächsten Projektgruppe im Kapitel Ausblick genannt. Die Dokumentation endet mit einem Fazit, in der eine persönliche Bewertung von allen Gruppenmitgliedern zu dem Projekt stattfindet.

Der Anhang enthält eine Installation- und Bedienungsanleitung der einzelnen Komponenten für eine Inbetriebnahme des gesamten Systems.

2 Aufgabenstellung

In unserem Projekt ging es darum, die Erfassung und Auswertung von Schwimmwettkampfergebnissen zu automatisieren. In der Ausgangssituation werden die Wettkampfergebnisse bei Wettkämpfen mit Handzeitnahme handschriftlich erfasst und ausgewertet. Das ist bei wechselnden Wettkämpfen im Minutentakt ein äußerst stressiges Unterfangen.

Vor allem die Übergabe der handschriftlichen Ergebnisse von den Zielrichtern und den Zeitnehmern an den Auswerter sowie die Erfassung der handschriftlichen ausgewerteten Ergebnisse am Rechner sollen dabei eingespart werden. Außerdem soll die Arbeit in der Auswertung wesentlich erleichtert werden. Im Normalfall gibt es drei bis fünf Zielrichter, die bei jedem Schwimmlauf lediglich die Reihenfolge des Einlaufs der Schwimmer erfassen. Das heißt, sie schreiben die Bahnnummern derjenigen Schwimmer in der Reihenfolge auf, in der sie deren Zielanschlag gesehen bzw. wahrgenommen haben. Es gibt immer eine ungerade Anzahl an Zielrichtern, damit im Notfall immer ein Mehrheitsentscheid möglich ist. Für den Fall von unklaren Situationen nimmt auch der Schiedsrichter den Zieleinlauf auf, der vom Auswerter erfragt werden kann.

Unsere Aufgabe war es jetzt, die in jedem Lauf mit den Positionen beschrifteten Zettel durch Smartphones zu ersetzen. Hier soll es dann möglich sein, die Bahnen per Drag & Drop den jeweiligen Positionen zuzuordnen.

Bei den Zeitnehmern soll die Stoppuhr durch einen „Drücker“ ersetzt werden, der ähnlich wie bei einer Stoppuhr bei Zwischen- und Endzeiten betätigt werden soll. Das Drücken beim Start entfällt hiermit dadurch, dass die mit den Drückern verbundenen „Kleincomputer“ direkt mit dem elektronischen Startsignal der bereits vorhandenen Startanlage gespeist werden.

Zusätzlich soll ein PC mit einer kontrollierenden Funktionalität verwendet werden. Dieser empfängt nach jedem Wettkampf das Resultat der Mini-PCs. Auf diesem soll zu sehen sein, welcher Lauf gerade stattfindet, sowie welche bereits abgeschlossen sind und noch folgen. Außerdem soll es hier möglich sein, Wettkämpfe vorzuziehen, zu beenden und auf Fehlstarts zu reagieren. Er stellt damit eine Wettkampfübersicht und Kontrollinstanz dar. Zusätzlich werden vom Kontroll-PC die Platzierungen der

Zielrichter-App entgegengenommen. Nach dem Eintreffen sämtlicher Daten werden alle benötigen Daten zur weiteren Verarbeitung an einen Auswerter-PC weitergeleitet.

Mit dem Auswerter-PC wird kontrolliert, ob die Reihenfolge der Zieleinläufe (Zielrichter) mit den genommenen Zeiten übereinstimmt. Bei Diskrepanzen wird dem Auswerter (Aufgabe im Kampfgericht) ein Vorschlag zur Zeitkorrektur unterbreitet, den er übernehmen oder manuell nachbearbeiten kann. Die ausgewerteten Daten werden letztendlich an die Software „EasyWK“ übergeben, um das Protokoll erstellen und Urkunden drucken zu können.

3 Projektmanagement

Eine weitere wichtige Aufgabe des Projekts war ebenfalls die Anwendung erlernter Methoden bezüglich des Projektmanagements.

Zu Beginn des Projekts gab es eine zweiwöchige Intensivphase, in der zunächst jeder Teilnehmer der Projektgruppe ein bestimmtes Thema vorbereitet und es dann den anderen vorgestellt hat. Dadurch sollte der Einstieg in das Projekt erleichtert werden und bereits erste Diskussionen für die weitere Vorgehensweise ausgelöst werden. So wurde dann auch in dieser Phase der vorläufige Projektleiter gewählt und besprochen, nach welcher Methode das Projektmanagement ausgelegt wird.

Die Gruppe hat sich dann als Projektmanagement Methode auf eine Mischung aus *Scrum* und *Extreme Programming* festgelegt. Dabei sollte es jede Woche eine Besprechung in Form eines *Weekly Scrums* geben, um alle Teilnehmer immer auf den aktuellsten Stand zu bringen. Bei der Methode *Scrum* ist es üblich, jeden Tag ein kleines Meeting durchzuführen. Da dies aber zeitlich nicht mit den einzelnen Teilnehmern vereinbar war, wurde sich auf wöchentliche Meetings geeinigt und eine *WhatsApp* Gruppe eingerichtet, um aktuelle Informationen und Probleme zu diskutieren.

Auch sollten bei unserer Mischung der Methoden pro Gruppe Meilensteine definiert werden, um die Zeitpunkte der Fertigstellung der einzelnen Projektphasen immer im Auge zu behalten. Somit entspricht unsere Methode einer agilen Vorgehensweise, aber trotzdem mit fixen Meilensteinen. Dadurch übernehmen wir die Flexibilität der agilen Vorgehensweisen, um die einzelnen Projektphasen nicht starr zu durchlaufen, aber auch die feste Terminplanung mit Meilensteinen, um den Zeitpunkt der Fertigstellung für den Schwimmwettkampf – das Adventsschwimmen des SV Neptun Emden am 7. und 8. Dezember 2018 – nicht außer Acht zu lassen.

Nach dieser Entscheidung wurde das gesamte Systemkonzept erstellt und dann in einzelne Komponenten unterteilt. Dann haben sich die Teilnehmer, je nach geschätztem Aufwand, in Gruppen für die Komponenten aufgeteilt. Festgelegt wurde dann auch, dass sich die Teilgruppen innerhalb ihrer Gruppe selbst organisieren.

Das gemeinsame Arbeiten an der Software der einzelnen Komponenten wurde durch das Versionsverwaltungstool *Gitlab* unterstützt. Dort wurden auch aktuelle Aufgaben der einzelnen Gruppen, die Meilensteine und allgemeine Themen, wie zum Beispiel die Abstimmung der Schnittstellen, eingetragen. So hatten auch alle Teilnehmer immer einen Überblick über ihre eigene Teilgruppe, über die anderen Gruppen und allgemeine festgelegte Themen. Kam es zwischendurch zu schwerwiegenden Problemen, wurde Rücksprache mit den Betreuern via E-Mail oder einem persönlichen Treffen gehalten.

Im Verlauf des Projekts hat sich jedoch herausgestellt, dass diese Vorgehensweise nicht die optimalste für unser Projekt und dessen Ziel ist. Daher wurde die Vorgehensweise des Projektmanagements etwas geändert und auch die Aufgabe der Projektleitung wurde an ein weiteres Teammitglied übergeben.

Nach dem Wechsel des Projektleiters gab es jede Woche zur Besprechung einen neuen Prototypen, der dann wöchentlich weitere Funktionalitäten erhielt. Zu Beginn der Besprechung wurde dieser Prototyp immer wieder getestet. Dadurch konnten öfters Fehler festgestellt und somit rechtzeitig ausgebessert werden.

Diese Vorgehensweise entspricht wieder dem Grundsatz von *Scrum*, bei dem nach jedem Sprint, also einem bestimmten Zeitabschnitt, bei dem neue Funktionalitäten hinzugefügt werden, ein erweiterter Prototyp vorgestellt wird. Bei uns dauerten die Sprints immer eine Woche.

Auch gab es mehrmals ein *Skype Meeting* mit immer mindestens einem Teilnehmer pro Teilgruppe, um weitere Vorgehensweisen und mögliche Schwierigkeiten zu klären. Später hat sich die Gruppe auch noch sehr oft zusätzlich ohne die Betreuer getroffen, um weitere Tests mit dem Gesamtsystem durchzuführen.

4 Systemkonzept

Nach der zweiwöchigen Intensivphase entstand ein erster Entwurf, wie das Endprodukt aussehen soll. Der Entwurf baute in vielen Teilen auf den Erkenntnissen der Projektgruppe 2017 auf, erweiterte diese aber auch.

4.1 Konzept der Projektgruppe 2017

Die Projektgruppe 2017 nutzte folgende Komponenten:

- Zielrichter erhielten jeweils ein Smartphone mit einer App. In der App konnte eingetragen werden in welcher Reihenfolge der Zieleinlauf der Schwimmer gesehen wurde.
- Für die Zeitnahme wurde ein einzelner Raspberry-Pi bereitgestellt. An diesem wurden acht Drücker über die GPIO-Schnittstelle angeschlossen. Die Startanlage wurde ebenfalls an diesen Raspberry-Pi angeschlossen.
- Um einen knappen Zieleinlauf besser nachvollziehen zu können, wurde auch ein Kamera-Tool entwickelt. Die Kamera sollte automatisiert die letzten Momente eines Wettkampfes aufnehmen, sodass man Zieleinläufe beliebig oft in Zeitlupe betrachten konnte.
- Der Mittelpunkt der gesamten Anwendung war der Auswerter-PC, genannt „Contest Master“. Alle anderen Komponenten haben Ihre Ergebnisse an den Auswerter geschickt, welcher die Daten dann verarbeitet hat und für den Export in ein EasyWK-Format zuständig war. Um die Daten persistent zu speichern wurde eine lokale Datenbank eingerichtet.

Diese Konstellation hatte einige Probleme. Die Drücker mussten alle an einer zentralen Stelle angeschlossen werden. In einem Hallenbad ist es sehr umständlich solch eine Menge Kabel am Beckenrand zu verlegen, ohne dass die Kabel zu einer Stolpergefahr werden. Außerdem würde die Zeitnahme aller Drücker ausfallen, sobald der zentrale Raspberry-Pi einen Fehler hat. Die zentrale Stelle des Auswerter-PCs ist ein weiteres Problem. Die Auswertung bei Schwimmwettkämpfen kann zum Teil sehr hektisch vonstattengehen, weswegen der Auswerter-PC in dieser Zeit nicht genutzt werden kann, um eventuelle Änderungen im Wettkampfverlauf vorzunehmen (z.B. ein Lauf wird mit einem anderen getauscht).

4.2 Erstes Gesamtkonzept

In der Intensivphase zu Anfang des Projekts hat man sich mit den oben genannten Punkten auseinandergesetzt und diese bewertet. Dabei wurde ein erstes Gesamtkonzept erstellt (Abbildung 1 Erstes Konzept).

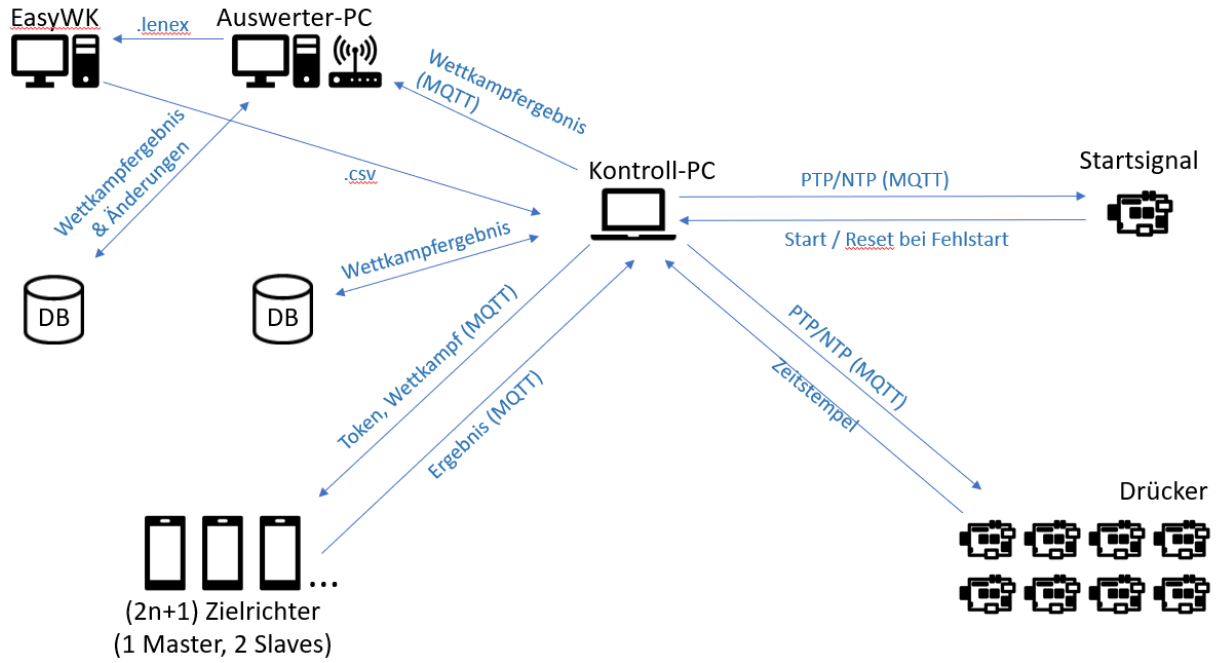


Abbildung 1 Erstes Konzept

- Die Erfassung der Zielrichterergebnisse blieb größtenteils unverändert. Die Eingabe erfolgt über eine Handy-App.
- Für die Drücker wurde nun ein Raspberry-Pi pro Drücker vorgesehen, nicht ein einziger Raspberry-Pi für alle Drücker. Dies sollte die Menge an Kabeln reduzieren und auch einen Single-Point-of-Failure ausschließen. Jeder Raspberry-Pi bekommt außerdem seinen eigenen Akku, um die Menge der Kabel bei der Stromversorgung ebenfalls einzusparen. Um Fehlern vorzubeugen, sollten die Drücker möglichst wenige Aufgaben haben, die Logik sollte im Kontroll-PC und Auswerter-PC gebündelt werden. Die Drücker schicken beim Tastendruck lediglich einen Timestamp an den Kontroll-PC.
- Das Startsignal wird von einem weiteren unabhängigen Raspberry-Pi entgegen genommen. Dieser soll direkt neben der Startanlage stehen. Sobald das Signal kommt, werden alle beteiligten Instanzen per MQTT über WLAN informiert.
- Der alte Auswerter-PC wurde in zwei Teile zerlegt. Es wurde zusätzlich der Kontroll-PC erschaffen. Die Aufgabe des Kontroll-PCs ist es, einen Echtzeitüberblick über den/die laufenden Wettkampf/Wettkämpfe zu liefern. Geplant war, dass hier keinerlei Eingaben notwendig sind. Es wird lediglich gezeigt welchen Status der aktuelle Lauf hat, welcher Lauf als nächstes kommt und die gesammelten Daten an den Auswerter-PC weiterzuleiten. Als Backup, im Falle eines Programmabsturzes, wird eine lokale Datenbank genutzt. Vor dem Wettkampf sollte der Kontroll-PC außerdem eine CSV-Datei mit dem Meldeergebnis des Wettkampfes erhalten. So ist bekannt, in welcher Reihenfolge die Wettkampfläufe stattfinden. Falls sich die Reihenfolge ändert, ist ein Tausch auch am Kontroll-PC möglich.
- Der neue Auswerter-PC kümmert sich nun nur noch um die Auswertung. Die einzige Aufgabe des Auswerter-PCs besteht darin, die Ergebnisse vom Kontroll-PC zu empfangen und diese sinnvoll zu bearbeiten. Die Abstimmung der Zielrichter, welche Bahn auf welcher Position ist,

sollte dargestellt werden. Außerdem auch alle End- und Zwischenzeiten für jede Bahn. Sollte es Widersprüche zwischen der Zielrichter-Abstimmung und den Zeiten geben, sollte automatisch ein Mittelwert gebildet werden. Das Endergebnis (amtliches Ergebnis) stellt einen Vorschlag für den Auswerter dar und soll ohne Einschränkung manuell änderbar sein. Die ausgewerteten Wettkampfdaten werden in die Software EasyWK portiert über ein spezielles LENEX-Format.

4.3 Zweites Gesamtkonzept

Nach der anfänglichen Intensivphase begannen die Arbeiten an dem Projekt. Alle Gruppenmitglieder haben sich in ihre jeweiligen Themengebiete eingearbeitet. Nach einigen Wochen wurde bei vielen Gruppen festgestellt, dass das ursprüngliche Konzept einige Probleme hat, bzw. dass manche Problemstellungen auf andere Weise besser gelöst werden könnten. Das Gesamtkonzept wurde daraufhin neu aufgerollt und abgeändert. (Abbildung 2 Zweites Konzept)

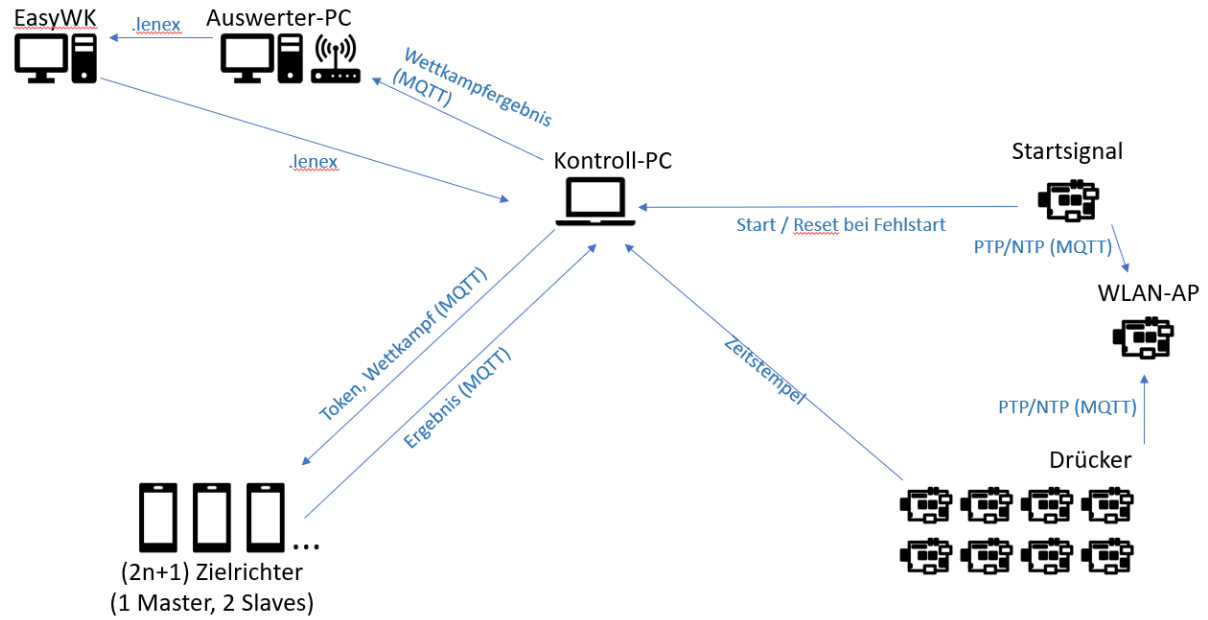


Abbildung 2 Zweites Konzept

- Der Kontroll-PC sollte ursprünglich als PTP/NTP Master für die Rasperry-Pis dienen. Dies stellte sich allerdings als problematisch raus, da der Kontroll-PC für Windows entwickelt wurde und sich kein geeigneter PTP/NTP Dienst fand. Man hat sich dafür entschieden den WLAN-Accesspoint daher um diese Funktion zu erweitern.
- Die lokalen Datenbanken des Auswerter- und Kontroll-PCs wurde nochmals betrachtet und evaluiert. Das Ziel war es ursprünglich, die Vorzüge einer Datenbank beim persistenten Speichern der Daten zu nutzen. Da die Menge an zu speichernden Daten aber äußerst überschaubar war, und die Einrichtung und Integration der Datenbanken komplizierter als anfangs geplant verlief, entschied man sich dazu, die zu speichernden Daten einfach als Textdatei abzuspeichern. Diese Entscheidung fiel auch vor dem Hintergrund, dass eine Datenbank, welche auf demselben Speichermedium wie das Programm selbst gespeichert wird, in keiner Weise eine bessere Ausfallsicherheit bietet. Somit wurde die Datenbank restlos gestrichen. (Siehe 6.4 Datenbank)
- Auch sollten Überkopfstarts, ein neuer Lauf beginnt, während ein anderer noch nicht vollständig beendet ist, berücksichtigt werden.

Das Zweite Konzept stellt den Ablauf des beim Adventsschwimmen 2018 in der Emder Friesentherme eingesetzten Produktes sehr nahe dar, es wurden danach keinerlei Änderungen, die den groben Ablauf betreffen, mehr vorgenommen. Der Kontroll-PC erhält zuerst eine Lenex-Datei aus EasyWK mit dem Meldeergebnis. So ist dem Kontroll-PC bekannt, welche Wettkämpfe stattfinden, in welcher Reihenfolge und welche Schwimmer teilnehmen. Der aktuelle Wettkampf wird auch der Zielrichter-App mitgeteilt. Der Startsignal-Raspberry-Pi schickt beim Start des Wettkampfs ein Signal an den Kontroll-PC, sodass dieser weiß zu welcher Uhrzeit der Lauf gestartet ist. Sobald einer der Drücker auslöst, schicken diese ihren derzeitigen Timestamp an den Kontroll-PC. Der Kontroll-PC kann den Timestamp dann mit der Uhrzeit des Startsignals verrechnen und eine Zwischen-, bzw. Endzeit bilden. Die Zielrichter sind in der Regel die letzte Instanz, da diese erst ihr Ergebnis abgeben sollten, sobald alle Zeiten genommen wurden und somit alle Schwimmer ins Ziel geschwommen sind. Sobald die Zielrichter ihre Abstimmung abgeben, gilt der Lauf als beendet und die Daten werden vom Kontroll-PC an den Auswerter-PC geschickt. Auch Überkopfstarts wurden implementiert, indem ein alter Lauf erst nach einer konfigurierbaren Anzahl von Sekunden beendet, nachdem das Startsignal für den neuen Lauf gegeben wurde. Dort wird der Lauf ausgewertet und letztendlich in Lenex exportiert, welches in EasyWK importiert wird.

5 Modellierung

5.1 Netzwerk

5.1.1 Aufbau

Die Komponenten des Netzwerks sind:

- Ein Raspberry Pi 3
- Beliebig viele Raspberry Pi Zero W, abhängig von der Anzahl der gewünschten Bahnen + einen zum Abgreifen des Startsignals
- Eine ungerade Anzahl an Smartphones, mindestens drei, maximal neun
- Der Kontroll-PC
- Der Auswerter-PC

Im Zentrum des Netzwerks steht ein Raspberry Pi 3 Model B+ mit einem angeschlossenen USB WLAN-Adapter. Dieser stellt einen Access-Point bereit, mit dem sich alle anderen benötigten Geräte verbinden können. Somit sind alle Geräte in einem Netzwerk und können untereinander kommunizieren.

Die Raspberry Pi Zero W‘s sollen Zeitstempel nehmen und an den Kontroll-PC weiterleiten. Die Pis nutzen ihren eingebauten WLAN Chip und bekommen alle jeweils eine feste IP zugewiesen, damit notfalls über eine SSH Verbindung auf das Betriebssystem zugegriffen werden kann.

Über eine App auf den Smartphones können die Zielrichter die Einlaufreihenfolge der Schwimmer eintragen, welche ebenfalls an den Kontroll-PC geschickt wird. Die Smartphones bekommen von unserem Access-Point über DHCP eine dynamische IP zugewiesen.

Der Kontrollrechner empfängt die Daten, berechnet die Zeiten und schickt am Ende eines Laufes gebündelt alle Daten an den Auswerter-PC weiter. Er stellt ebenfalls den MQTT-Broker zur Verfügung. Er bekommt eine feste IP zugewiesen, da allen Geräten, welche an der Kommunikation über MQTT teilnehmen möchten, diese Adresse bekannt sein muss.

Am Auswerter-PC können bei Unstimmigkeiten der Wettkampfdaten Korrekturen an diesen vorgenommen werden. Schlussendlich werden alle Daten ins Lenex-Format konvertiert und der EasyWK Software bereitgestellt. Er erhält ebenfalls eine dynamische Adresse.

5.1.2 MQTT

MQTT steht für Message Queuing Telemetry Transport. Das Protokoll ist ausgelegt für Machine-To-Machine Kommunikation. Die Funktionsweise beruht auf dem publish/subscribe Prinzip, bei dem es eine vermittelnde Instanz gibt, den sogenannten Broker, und beliebig viele weitere Geräte, welche alle jeweils Publisher und/oder Subscriber sein können.

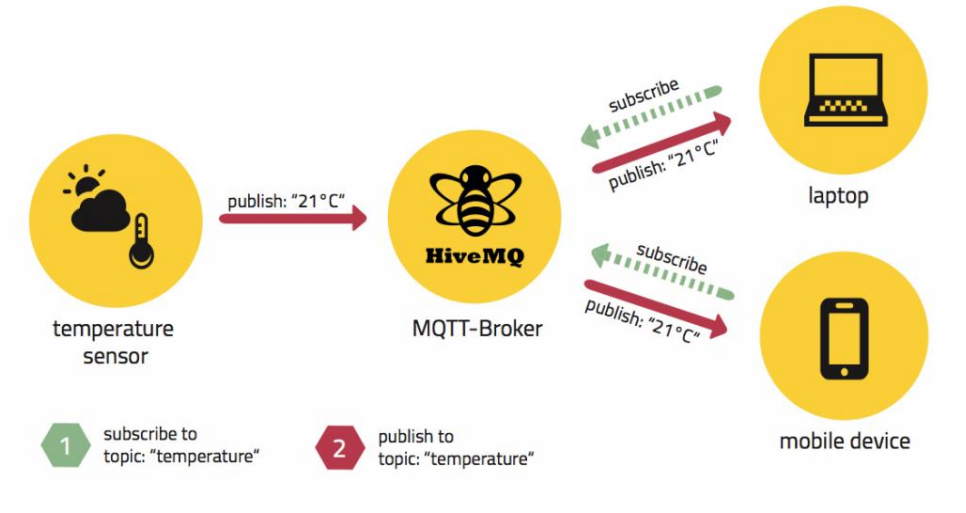


Abbildung 3 Funktionsweise MQTT anhand eines allgemeinen Beispiels (Abbildung aus [1])

In diesem Beispiel (siehe Abbildung 3) gibt es einen Laptop und ein Smartphone, welche beide beim Broker das Topic „temperature“ abonnieren.

Nun kommt ein Temperatur Sensor hinzu, der auf das Topic „temperature“ veröffentlicht. Das bedeutet, dass er seine gemessenen Temperaturen an den Broker schickt. Dieser überprüft nun, wer diese Temperaturdaten erhalten möchte und leitet sie weiter.

Wir haben uns für MQTT entschieden, da es eine sehr einfache Möglichkeit ist, Daten unter mehreren Teilnehmern in einem Netzwerk auszutauschen und da nur eine sehr geringe Bandbreite nötig ist. Ebenfalls müssen alle Geräte nicht die gesamte Topologie des Netzes kennen, sondern nur den Weg zum Broker. Dies würde es beispielsweise erleichtern, weitere Smartphones in das Netzwerk einzubinden.

5.2 Zeitmesssystem

Das Zeitmesssystem soll die Stoppuhren der Zeitnehmer bei einem Schwimmwettkampf ersetzen und die Übertragung der gestoppten Zeiten automatisieren.

Die Idee war es, die Drücker so simpel wie möglich zu halten, dies geschieht dadurch, dass beispielsweise die Zeiten nicht auf den einzelnen Geräten berechnet werden, sondern diese lediglich Zeitstempel von der globalen Zeit weiterschicken, welche dann durch Subtraktion die relative Zeit ergeben.

Auch die Bedienung sollte so intuitiv wie möglich erfolgen, daher gibt es nur einen einzigen Knopf, der sowohl dazu dient Zwischen-, als auch Endzeiten zu stoppen. Für eine weitere Funktionalität, ohne die Hardware zu verändern, könnte auch ein Gedrückthalten des Knopfes abgefragt werden.

Geschrieben wurde das Programm in der Programmiersprache C. Der Aufbau des Programms wird in Abschnitt 6.2.1 erläutert.

Die Raspberry Pi's nutzen zur Zeitsynchronisation untereinander PTP und als Zeitgeber den Kontrollrechner, der sich über NTP mit dem PTP-Master synchronisiert.

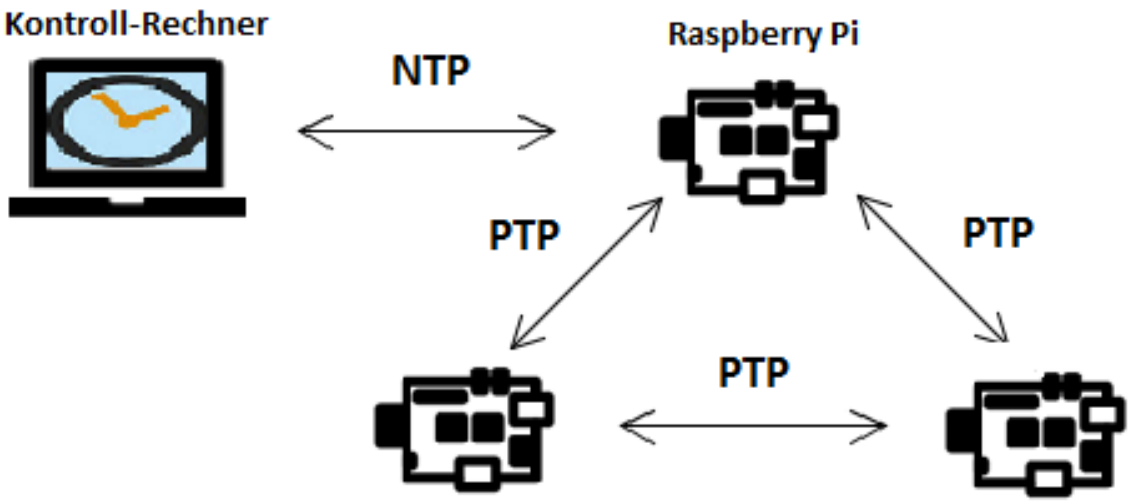


Abbildung 4 Zeitsynchronisation Raspberry Pi

5.3 Zielrichter-App

Die Zielrichter-App soll im Wesentlichen, die für den automatischen Auswerteprozess benötigte elektronische Erfassung der Zielrichterentscheide realisieren und dabei die manuelle Übermittlung der handschriftlichen Entscheide zum Auswerter überflüssig machen.

Die Zielrichter schreiben ihre gesehenen Zieleinläufe zunächst in das Meldeergebnis, das als Kladde für ein blindes Schreiben verwendet wird, um den Blick auf dem Zieleinlauf zu belassen. Aktuell folgt anschließend die Übertragung auf einen Kampfrichterzettel. Sämtliche Zettel der Zielrichter müssen dann eingesammelt und zum Auswerter gebracht und manuell ausgewertet werden. Durch die App sollen diese Schritte komplett entfallen und die Zielrichter geben ihre Ergebnisse mit Hilfe der App ab (durch die Übertragung aus den Notizen im Meldeergebnis), welche diese auch direkt an die weiteren Programme zur Kontrolle und darüber auch zur Auswertung übermittelt. Dadurch werden weniger Personen benötigt, es muss weniger Aufwand betrieben werden, die Ergebnisse liegen zur Auswertung direkt in digitalisierter Form vor und zusätzlich wird die Umwelt geschont, da weniger Papier verwendet wird.

Zur Modellierung der App für die Zielrichter wurde als Vorlage die App der Projektgruppe vom Wintersemester 2017/18 verwendet (siehe [4]).

Zu Beginn des Projekts wurde in der Intensivphase die Zielrichter-App der vorherigen Projektgruppe durch eine Teilnehmerin analysiert und der Gruppe vorgestellt. Aufgrund dieser Vorlage mit allen Screenshots und den benötigten Aktivitäten, hat sich die Gruppe darauf geeinigt, die ganze Grundidee der Layouts zu übernehmen, sodass die Screenshots der vorherigen Gruppe als Grundlage für die Modellierung der Zielrichter-App dienten. In Abbildung 5 Screenshots Zielrichter-App der vorherigen Gruppe“ ist eine kleine Auswahl der Screenshots der vorherigen Gruppe zu sehen(siehe [4]).

Diese Screenshots dienten lediglich zur Orientierung für die Erstellung der Layouts der App.

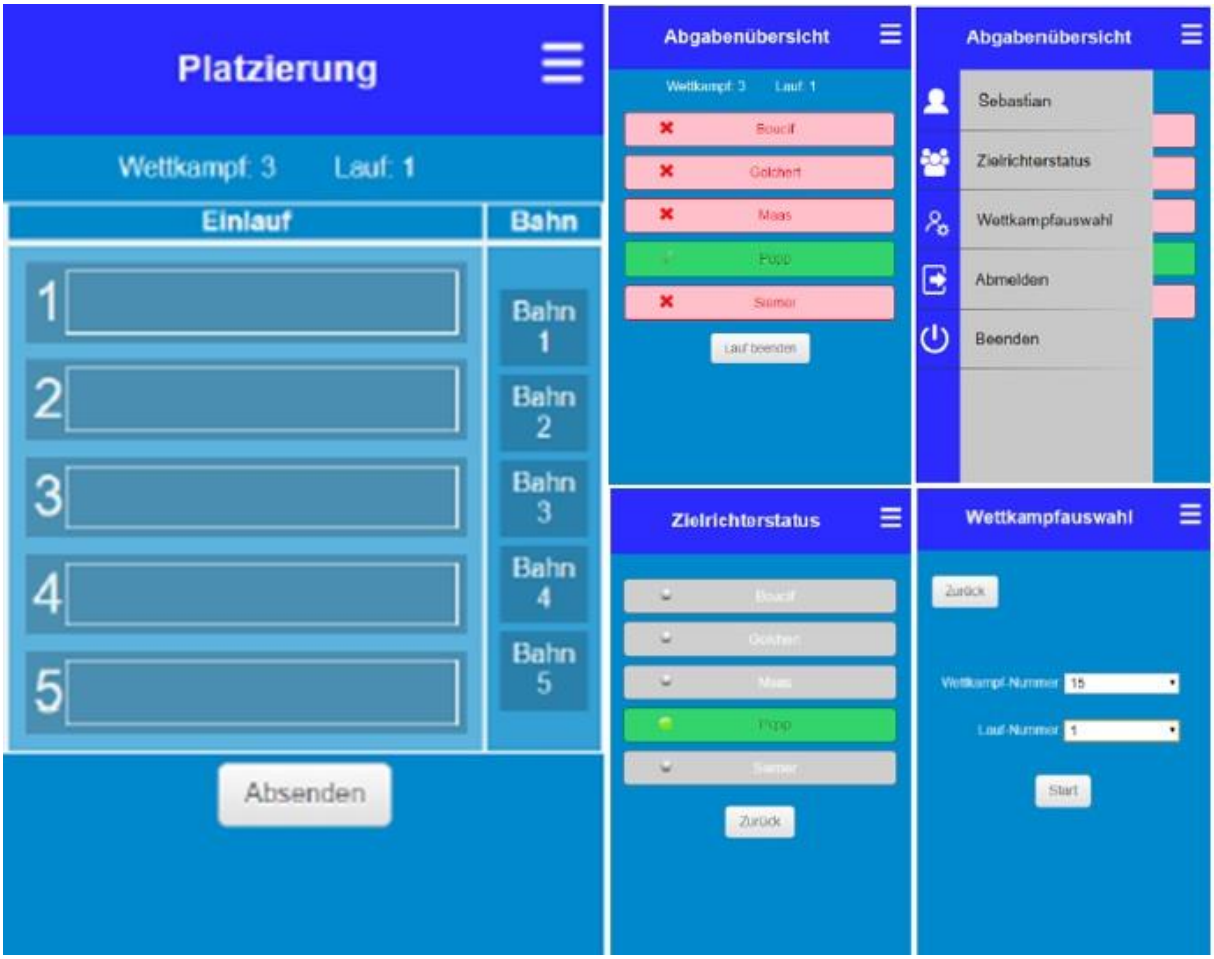


Abbildung 5 Screenshots Zielrichter-App der vorherigen Gruppe ([4])

Wichtig zu erwähnen ist noch, dass die Hauptfunktionalität der App darin bestehen soll, dass die Zielrichter ihre gesehenen Ergebnisse durch intuitives Drag and Drop abgeben können und die Ergebnisse dann auch direkt weiter verschickt werden. Dabei soll es möglichst einfach und schnell gehen die Ergebnisse abzusenden, da auf einem Schwimmwettkampf teilweise auch sehr schnell hintereinander die Läufe starten und die Zielrichter nicht viel Zeit haben, einen langen Dialog zu durchlaufen. Die Usability spielt an dieser Stelle eine besonders wichtige Rolle.

5.4 Datenbank

Hinweis: Die Datenbank war ursprünglich geplant und hat sich im Verlauf des Projektes als impraktikabel erwiesen. Weiteres zur Entscheidung finden Sie im Unterkapitel 6.4.2. Verwurf der Datenbank.

5.4.1. Aufgabe der Datenbank

Die Aufgabe der Datenbank ist die Speicherung von Wettkampfergebnissen und Ereignissen, die innerhalb des Systems entstehen. Es befindet sich (unmittelbar vor Entscheidung zur Revidierung der Datenbank) jeweils eine Datenbank auf dem Kontrollrechner und dem Auswerterrechner. Es gilt hierbei folgende mathematische Assoziation:

$$DB_{Kontrollrechner} \subseteq DB_{Auswerter}$$

Dies hat zur Folge, dass die Datenbank vom Kontrollrechner nach Durchlauf aller Wettkämpfe zum Auswerterrechner via MQTT-Broker gesendet und mit weiterführenden Informationen ergänzt wird. Alle Records vom Kontrollrechner bleiben somit bestehen. Heißt nun im Klartext, dass einzelne

Verarbeitungsschritte über die Datenbank verfolgbar sind. Die Datenbank erfüllt in dem **Sinne lediglich eine Backup-Funktion**, sofern der Kontrollrechner oder der Auswerter-PC crasht.

5.4.1. Auswahl des Datenbankmanagementsystems

Die Auswahl des Datenbankmanagementsystems ist entscheidend was folgende Punkte betrifft:

- Installations- und Konfigurationsaufwand,
- Managementaufwand,
- Programmierstil,
- Relational vs. Graphenbasiert vs. Key-Value-Store ➔ Umgang mit unseren Daten,
- Bei Relationalen Datenbanken: Welche Dialekte verwendet unser SQL-Code / Welche Features werden benötigt?

Für dieses Projekt kam eine gängige, relationale Datenbank in Frage. Es wurden hierbei Kompetenzen aus vorherigen Veranstaltungen gewonnen. Mittels der Sprache Structured Query Language (kurz SQL) ist es möglich, schnell und gezielt Daten und Informationen aus der Datenbank abzufragen und zu manipulieren. SQL ist im Vergleich zu Sprachen, wie C/C++ oder Java, keine imperative Programmiersprache, sondern eine deklarative. Im Klartext: In C/C++ schreibt der Entwickler den Code, *wie* etwas passiert, während bei SQL der Entwickler das, *was* er erreichen möchte, beschreibt.

Alle Softwareaspekte des Projektes (mit Ausnahme der Raspberry Pi-Drücker) sind in Java geschrieben. Ein Server für die Bereitstellung der Datenbank wäre für das Projekt überdimensioniert. Die Embedded-Variante empfiehlt sich eher als die Bereitstellung einer eigenen Serverinstanz in der Topologie des Systems. Unsere Datenbank sollte weitestgehend leichtgewichtig sein.

Respektive aller der genannten Punkte empfahl sich H2 ([7]), da es sich hierbei um ein in Java entwickeltes DBMS handelt. Der größte Vorteil von H2 ist die Vielzahl an Möglichkeiten, die Datenbank innerhalb einer Java-Applikation zu integrieren. Es existiert ein Servermodus, was die Datenbank innerhalb eines freigegebenen Bereichs zugreifbar macht. Die Datenbank kann zusätzlich – wie bei SQLite – direkt in eine Datei geschrieben werden. Der wohl interessanteste Punkt von H2 ist die Möglichkeit die Datenbank im RAM anzulegen. Es ist kein Geheimnis, dass der Arbeitsspeicher eine weitaus schnellere Möglichkeit ist als ein ordinärer Festplattenspeicher. Operationen in der Datenbank sind möglich, da die Menge an Daten nicht all zu groß ist, als dass diese nicht in den Arbeitsspeicher passen. Durch die Verwendung des Arbeitsspeicheransatzes besteht jedoch die Gefahr von Datenverlusten durch einen Crash. Hierbei hilft der hybride Modus, welcher H2 beinhaltet. In diesem Modus läuft die Datenbank weiterhin innerhalb des Arbeitsspeichers, jedoch wird zusätzlich eine Datei verwendet, in welcher die Datenbank zusätzlich repliziert wird. Der minimale Datenverlust und der Performancevorteil durch den Arbeitsspeicher sind die großen Vorteile des Verfahrens. Dennoch können Inkonsistenzen entstehen, wenn der Crash zwischen Aktualisierung der RAM-Datenbank und das Schreiben der Festplatten-Datenbank auftaucht.

5.4.3. Datenbankstruktur

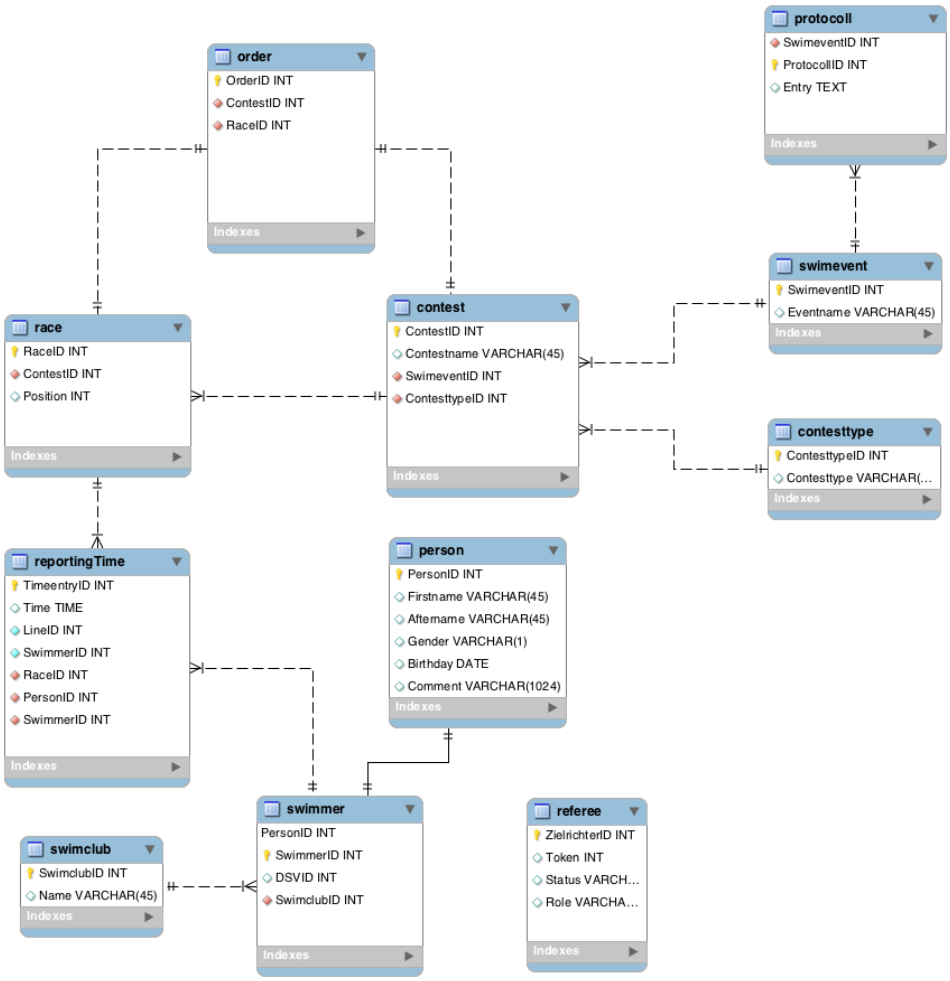


Abbildung 6: Eine in MySQL-Workbench angefertigte .erm-Datei, welche direkt als Datenbank transformiert werden kann. Aus dieser Relation heraus kann der JPA-Generator seine Entity-Klassen erstellen.

Die Datenbank ist gemäß der Abbildung 3 aufgebaut. Es existieren Personen, die Daten, wie Vor- und Nachnamen, Geschlecht, Geburtstag und zusätzliche Kommentare beinhalten. Personen können ebenfalls Schwimmer sein, welche zusätzlich eine Schwimmer-ID, eine DSVID (hierbei handelt es sich um eine eindeutige Registrierungsnummer des Deutschen Schwimmverbands DSV) und einem Schwimmclub zugehören. Schwimmer können mehrere ReportingTimes erhalten – diese erhalten die Zeiten eines Schwimmers im Kontext auf die Läufe auf eine bestimmte Bahn. Ein Lauf (race) sind einem direkten Wettkampf angeordnet und haben zusätzlich eine Position. Ein Wettkampf kann mehrere Läufe beinhalten. Wettkämpfe haben einen Namen und eine Wettkampfart. Es existieren verschiedene Arten von Wettkämpfen mit unterschiedlichen Meterangaben und einer Geschlechterdifferenzierung. Dem ganzen wird ein Schwimmevent angeordnet, das kann z.B. das Weihnachtsschwimmen 2018 sein. Eine Relation Log beinhaltet das Protokoll des Kontrollrechners, ebenso sind in dem Schwimmevent Zielrichter vorhanden. Diese Relation beinhaltet für jeden Zielrichter einen Token, den Status und seine Rolle.

5.5 Lenex Austausch

5.5.1. Allgemein

Der Lenex Parser ist wesentlicher Bestandteil des Kontroll- und Auswerter-PCs und ist für die Interpretation der Meldeergebnisdaten vom Typ Lenex (.lef) zuständig. Ebenfalls soll der Lenex Parser die finalen Ergebnisse in eine Datei schreiben, welche von EasyWK ([8]) eingelesen werden kann.

Grob formuliert enthält Lenex sämtliche Informationen über die anstehenden Wettkämpfe, Läufe, Schwimmclubs und Schwimmer [11]. Der Kontrollrechner benötigt die Meldeergebnisse, um die Daten eines Wettkampfes innerhalb unseres Systems zu importieren.

Nachdem der Kontrollrechner jeweils einen Wettkampflauf abgeschlossen hat, wird dieser (inklusive der erzielten Zwischen- und Endzeiten sowie der Zielrichterentscheide) an den Auswerter-PC weitergeleitet. Dort werden aus den Endzeiten und Zielrichterentscheiden die „amtlichen" Zeiten und Platzierungen festgelegt. Danach werden die Daten in eine Lenex-Datei geschrieben – diese kann über EasyWK mittels einer ALGE-Simulation (Zeitnehmersystem) eingelesen werden. Diese wird lediglich simuliert, da der direkte Import einer Lenex-Datei nicht möglich ist.

```
<LENEX version="3.0">
  <CONSTRUCTOR name="EasyWk" version="5.11" registration="">
    <CONTACT name="Bjoern Stickan" internet="http://www.easywk.de" email="info@easywk.de" />
  </CONSTRUCTOR>
  <MEETS>
    <MEET city="Emden" course="SCM" name="51. Internationales Adventsschwimmfest" nation="GER" organizer="SV Neptun Emden"
    hostclub="SV Neptun Emden" deadline="2018-11-23">
      <CONTACT city="Emden" email="meldungen@neptun-emden.de" fax="04921 9239501" internet="www.neptun-emden.de"
      name="Dr&#xFC;ner, Matthias" phone="04921 9239500" street="Franz-Schubert-Str. 18" zip="26721" />
      <AGEDATE type="YEAR" value="2018-01-01" />
      <SESSIONS>
        <SESSION number="1" date="2018-12-07" daytime="16:30" officialmeeting="16:00" warmupfrom="16:00">
          <EVENTS>
            <EVENT eventid="3" number="1" gender="F" round="TIM">
              <SWIMSTYLE stroke="MEDLEY" relaycount="1" distance="400" />
              <FEE value="500" currency="EUR" />
              <HEATS>
                <HEAT heatid="3001" number="1" daytime="16:30" />
                <HEAT heatid="3002" number="2" daytime="16:38" />
                <HEAT heatid="3003" number="3" daytime="16:45" />
              </HEATS>
```

Abbildung 7: Ausschnitt der Lenex-Datei (Header) vom 51. Internationales Adventsschwimmen.

In der Abbildung 7 ist zu erkennen, wie die von EasyWk generierte Lenex-Datei aufgebaut ist. Unschwer zu erkennen ist, dass Lenex auf XML basiert, welche für eine einheitliche Baumstruktur bekannt ist.

Den Rootknoten bildet <LENEX>. <CONSTRUCTOR> enthält lediglich die Erstellerinformationen der Lenexdatei (was ab diesem Punkt vernachlässigt werden kann). Die <MEETS>-Struktur enthält die relevanten Informationen. Jeder <MEET>-Tag leitet eine Veranstaltung ein. Eine Veranstaltung kann in mehrere Abschnitte unterteilt werden, die auf mehrere Zeitpunkte (z.B. Vormittag/Nachmittag) und Tage verteilt werden können – in XML wird ein Abschnitt des Tags <SESSION> eingeleitet. Einem Abschnitt werden grundsätzlich einige Properties zugewiesen:

- Die Abschnittsnummer (number),
- das Datum (date),
- die Einlasszeit,
- die Zeit des Beginns des Einschwimmens und
- die Zeit des Wettkampfbeginns.

Jede Veranstaltung enthält Wettkämpfe (<EVENTS>), die wiederum in mehrere Läufe (<HEATS>) strukturiert sind.

Ein Wettkampf <EVENT> besteht aus einer Event-ID und einer Wettkampfnummer (eventnumber), welche unabhängig voneinander sind. Während die Wettkampfnummer relevant für die Wettkampfveranstaltung ist, ist die Event-ID lediglich für die Kommunikation innerhalb von Lenex und EasyWk von Bedeutung. Folgende weitere Properties gehören zum Wettkampf (event):

- das Geschlecht (gender),
- Art des Wettkampfs (Vorlauf, Zwischenlauf, Entscheidung – round),
- die Schwimmdefinition (SWIMSTYLE) mit

    - der Schwimmdisziplin (stroke),
    - der Anzahl Staffelteilnehmer (relaycount – bei Einzelwettkämpfen 1),
    - der Streckenlänge (distance),
- den Meldegebühren (FEE) und
- den Läufen (HEAT).

In <HEAT> (Lauf) steht die Heat-ID und die Laufnummer (number) – ebenfalls gilt hier der gleiche Hinweis, wie bei Event (die Laufnummer ist relevant für die Wettkampfveranstaltung) – und die Uhrzeit, wann der Lauf voraussichtlich gestartet wird.

```
<CLUBS>
  <CLUB name="Azuro" nation="NED" region="0" code="0">
    <ATHLETES>
      <ATHLETE athleteid="20" birthdate="2001-01-01" gender="F" lastname="Wolterman" firstname="Kimberly"
      license="200103010">
        <ENTRIES>
          <ENTRY eventid="35" entrytime="00:00:46.97" heatid="35011" lane="3" />
          <ENTRY eventid="33" entrytime="00:01:21.13" heatid="33010" lane="4" />
          <ENTRY eventid="27" entrytime="00:00:40.81" heatid="27014" lane="1" />
          <ENTRY eventid="19" entrytime="00:01:34.01" heatid="19010" lane="2" />
          <ENTRY eventid="11" entrytime="00:00:36.88" heatid="11015" lane="1" />
          <ENTRY eventid="7" entrytime="00:00:00.00" heatid="7001" lane="3" />
        </ENTRIES>
      </ATHLETE>
      <ATHLETE athleteid="21" birthdate="2002-01-01" gender="M" lastname="van der Ploeg" firstname="Harmen"
      license="200203695">
        <ENTRIES>
          <ENTRY eventid="38" entrytime="00:01:11.76" heatid="38005" lane="5" />
          <ENTRY eventid="36" entrytime="00:00:35.37" heatid="36010" lane="5" />
          <ENTRY eventid="34" entrytime="00:00:59.92" heatid="34011" lane="3" />
          <ENTRY eventid="28" entrytime="00:00:32.36" heatid="28010" lane="2" />
          <ENTRY eventid="20" entrytime="00:01:09.59" heatid="20007" lane="2" />
          <ENTRY eventid="18" entrytime="00:00:31.31" heatid="18004" lane="4" />
          <ENTRY eventid="12" entrytime="00:00:26.28" heatid="12015" lane="5" />
          <ENTRY eventid="8" entrytime="00:01:18.97" heatid="8008" lane="2" />
        </ENTRIES>
      </ATHLETE>
```

Abbildung 8: Ausschnitt der Lenex-Datei (Body) vom 51. Internationales Adventsschwimmen.

Innerhalb von <CLUBS> sind die Schwimmvereine samt den Schwimmern (<ATHLETE>) und dessen relevanten Schwimmdaten enthalten. Die Meldezeiten, die Event- und Lauf-ID (nicht die Nummern!) und Bahn, auf welcher der Schwimmer schwimmt, befinden sich in <ENTRY>.

5.5.2. Lenex zu Kontrollrechner

Der Parser für den Kontrollrechner besteht aus 2 größeren Komponenten:

1. XML-Inhalte zu POJO (Plain Old Java Object)
2. POJO zu den Models

Bei dem 1. Punkt ist JAXB eine gute Abhilfe, da mittels JAXB POJOs automatisch generiert werden. Die zu verwendete Lenexdatei wird dabei in einer .xsd-Datei konvertiert – dazu existieren haufenweise Tools im Netz. Diese Datei wird dann anschließend genutzt, um daraus die POJOs zu erstellen. Eclipse samt dem JAXB-Plugin bietet dazu die Möglichkeit, diese einfach in der GUI zu generieren.

Im 2. Schritt müssen nun die allgemeine POJOs in unserer Softwarearchitektur integriert werden. Diese Verantwortung übernimmt die Klasse LenexParser, wozu mehr im Implementierungskapitel zu finden ist.

5.5.3. Auswerter zu Lenex/EasyWk

Die am Auswerterrechner ausgewerteten Daten müssen in einem exakten Format streng nach einem Schema an EasyWk weitergegeben werden. Eine direkte Importierung von Lenex-Dateien ist in EasyWk (Stand: Dezember 2018) nicht möglich. Jedoch existiert der Umweg über die Simulierung einer ALGE-

Zeitmessanlage. Hierbei nutzen EasyWk und das ALGE-System einen gemeinsamen Ordner mit 2 Dateien.

- splash_send.txt:
  In dieser Datei schreibt EasyWk eine Anfrage an das ALGE-System. Die ALGE nutzt die Anfragen, um zu wissen, **welche Ergebnisdaten** benötigt werden.

  EasyWk sendet jede 8. Sekunde ein `VERSION;…` worauf die ALGE ebenfalls mit einem `VERSION` antwortet. Dieser bidirektionale Ping dient dazu, dass der Benutzer in EasyWk informiert ist, dass die Kommunikation zwischen EasyWk und der ALGE steht.
  Sofern in EasyWk der Button „Zeiten anfragen“ gedrückt wird, wird auch in splash_send folgende Zeile geschrieben:

  ```
  ASK RESULTS;EVENTID=<EventID>;HEATID=<HeatID>
  ```

  Wie zuvor beschrieben, wird innerhalb des Lenex-Formats mit den IDs gearbeitet und nicht mit den Nummern. Es wird also vom Event mit der EventID ein Lauf mit der HeatID angefragt. Die ALGE nutzt diese Zeile, um nun die erzielten (in unserem Fall amtlichen) Zeiten des Laufs zurückzuschicken.

- splash_receive.txt:
  die Antwort der ALGE wird nun in splash_receive geschrieben.
  Wenn *EasyWk einen Ping (via VERSION) an die ALGE schickt*, antwortet die ALGE ebenfalls mittels `VERSION`.
  Sofern *kein Lauf existiert*, wird lediglich `SEND RESULTS; NOT FOUND` geschrieben. Damit weiß EasyWk, dass keine Zeiten zu dem angefragten Lauf existieren und nicht weiter auf einen Lauf gewartet werden muss.
  Falls ein *Lauf gefunden wurde*, werden die Laufzeiten in einer speziellen Syntax angegeben, welche zur Importierung eingehalten werden muss.

  ```
  SEND RESULTS;START
  <HEAT heatid="<HeatID>">
  <RESULTS>
  <RESULT lane="<LaneNo>" swimtime="<Swimtime z.B. 00:02:05.00>"
  backuptime1="<Swimtime z.B. 00:02:05.00>" status="<Statusflag>">
  <SPLITS>
  <SPLIT distance="<Distance>" swimtime="<Zwischenzeit>" />
  …
  </SPLITS>
  </RESULT>
  …
  </RESULTS>
  </HEAT>
  SEND RESULTS;END
  ```

  Mittels SEND RESULTS;START wird die Datenübertragung gestartet, danach folgen die relevanten Daten die innerhalb von EasyWk aufgenommen werden sollen. Es wird ein Lauf angegeben, welche die HeatID beinhaltet. Auch hier gilt der Hinweis: LENEX arbeitet mit der ID, nicht mit der Nummer. Danach folgen die Ergebnisse mittels RESULTS. Die Ergebnisse werden einer Bahn zugewiesen, nicht einer Person. Großer Vorteil dieser Variante ist die leichtere Korrektur von Abweichungen der Meldedaten. Die Meldedaten sind i.d.R. nicht

äquivalent zu den Läufen, da Bahnplatzierungen geändert werden, Schwimmer nicht erscheinen oder kurzfristig andere Schwimmer an einem Lauf teilnehmen. Es passiert sogar, dass in 2 Läufen des gleichen Wettkampfs nur ein Bruchteil der Schwimmer teilnehmen. Da ist es naheliegend, dass die beiden Läufe zu einem zusammengefasst werden. In einem RESULT steht somit die Bahn-Nr., die Schwimmzeit und eine Backupzeit. Die Schwimmzeit kommt von der automatischen Zeitmessung, während sich bei der Backupzeit um eine manuelle Zeit handelt, welche ähnlich der von uns genommenen Zeit von einem Zeitnehmer bedienten Drücker ist. Ebenfalls muss ein Statusattribut angegeben werden. Sofern eine Zeit vorhanden ist, bleibt das Attribut jedoch leer. Andernfalls enthält der Status ein DNF – was für „did not finish“ steht. EasyWk benötigt theoretisch nicht die Zwischenzeiten. Da jedoch i.d.R. bei längeren Schwimmstrecken und bei Staffeln Zwischenzeiten aufgenommen werden, können diese mittels des SPLIT-Tag eingeleitet werden. Jede Zwischenzeit startet mit dem SPLIT-Tag und enthält eine distance – sprich die Distanz, an welcher die Zwischenzeit aufgenommen wurde – und die swimtime. Wenn alle Daten angegeben sind, wird mittels SEND RESULTS;END das Ende der zu sendenden Daten gekennzeichnet.

5.6 Kontrollprogramm

Das Kontrollprogramm ist das Herzstück des gesamten Systems. Die gesamte Kommunikation zu den einzelnen Komponenten wird darüber überwacht und gesteuert. Die Daten werden entgegengenommen, kontrolliert und verarbeitet.

Zu Beginn der Intensivphase wurde zuerst ein Mockup des Kontrollprogramm (siehe Abbildung 9) auf der Webseite www.draw.io erstellt. Daran konnten die Funktionalitäten und das Aussehen des Programms besser ermittelt werden. Das Mockup diente gerade in der Anfangsphase der Orientierung und es konnten so während der Besprechung im Team neue Vorschläge zum Design und zu Funktionalitäten mitgeteilt werden. Das Startfenster des Programms dient erstmal nur der Verbindung mit den Zielrichtern und dem Einlesen der Wettkampfdaten. Das Dashboard gibt dabei grob Aufschluss über den aktuellen Wettkampf, Lauf und der besetzten Bahnen. In den Einstellungen können dann nochmal die Verbindung zu den Zielrichtern und der Wettkampfdaten geprüft und geändert werden.

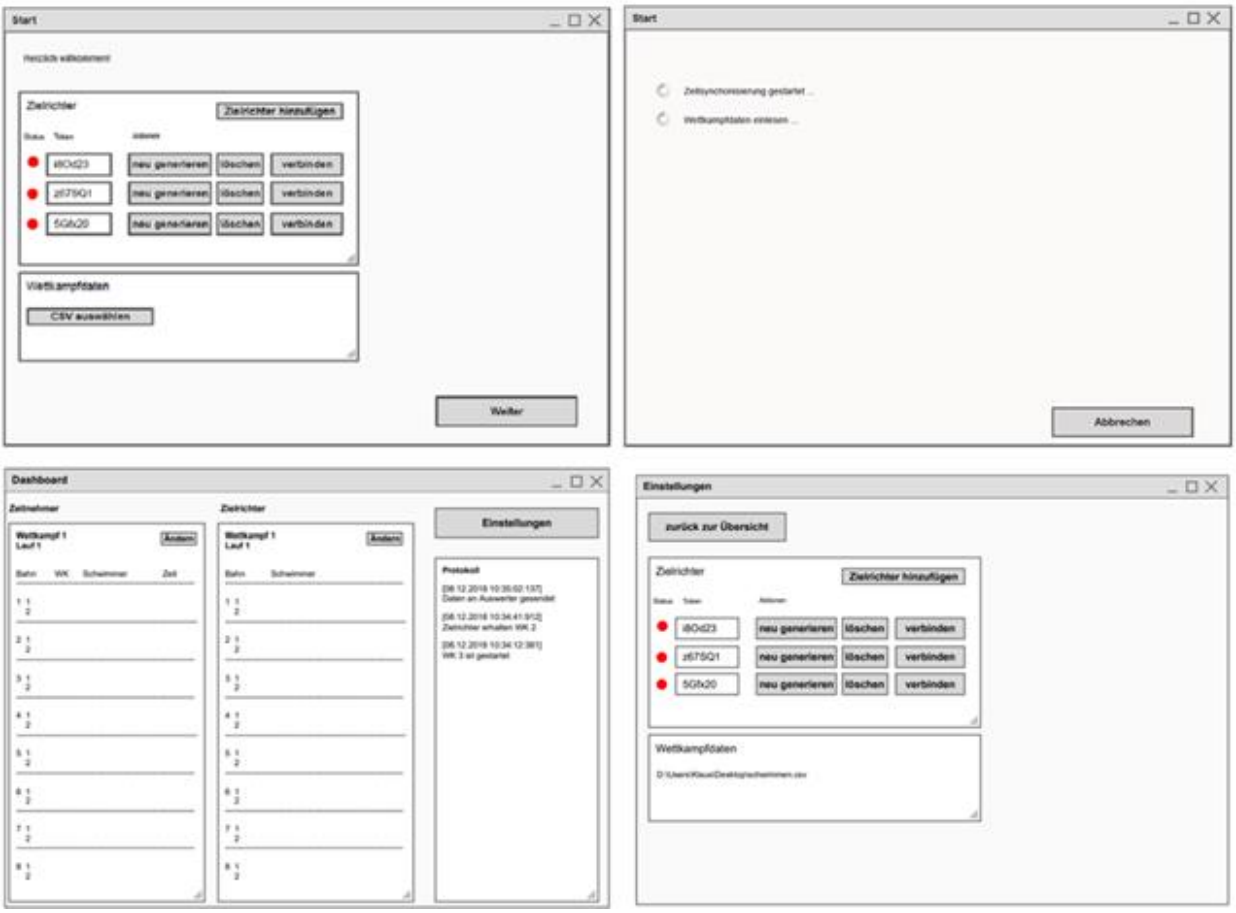


Abbildung 9 anfänglich erstelltes Mockup des Kontroll-Programms

Es wurde ebenfalls eine Liste der Anforderungen im GitLab erstellt. Diese konnte dann im Laufe des Projektes beliebig erweitert werden.

5.7 Auswertungsprogramm

5.7.1 Aufgabe des Auswerters

Der Auswerter-PC ist die letzte Komponente im Gesamtsystem, bevor Daten an die Software EasyWK übergeben werden. Seine Aufgabe ist es, alle beendeten Wettkämpfe vom Kontrollrechner anzunehmen, ggf. zu bearbeiten und dann in ein Lenex-Format zu konvertieren. Die Auswerter-App erhält diese Informationen per MQTT vom Kontrollrechner. Die Auswerter-App zeigt diese Daten in einer Übersicht an, in der für jede Platzierung der Jahrgang, alle Zielrichter-Abstimmungen und Zeiten angezeigt werden. Auf Basis dieser Daten, versucht das Programm einen Platzierungsvorschlag zu erstellen. Der Auswerter (eine Person des Kampfgerichts, der die Auswerter-App bedient) kann diese Daten übernehmen oder manuell modifizieren. Wenn die auszuwertenden Daten Widersprüche enthalten, werden diese dem Auswerter durch farbliche Markierungen kenntlich gemacht.

Sobald die Daten exportbereit sind, können sie exportiert werden. Die Lenexdatei kann dann in EasyWK geladen werden. Eine Instanz des EasyWK-Programms sollte sich hierfür idealerweise auf demselben Rechner befinden.

5.7.2 Datenaustausch mit Kontrollrechner

Um Daten verarbeiten zu können, musste schon früh im Projekt geklärt werden, wie die Daten zur Auswerter-App kommen. Aufgezeichnet werden die Informationen bei den Drückern sowie bei den Handys für die Zielrichter. Der Kontrollrechner sammelt diese Informationen unverarbeitet. Für die Auswerter-App ist also nur ein Austausch mit dem Kontrollrechner nötig. Eine andere Idee war, dass

alle Stationen ihre Informationen direkt an die Auswerter-App schicken, sodass dieser den Ablauf des Wettkampfs in Echtzeit darstellen kann. Aufgrund der Auswerte-Algorithmen war es aber am einfachsten, immer mit einem vollständigen Datensatz zu arbeiten. Deswegen wurde beschlossen, die Informationen beim Kontrollrechner zu sammeln und gebündelt an die Auswerter-App zu schicken. Ein weiterer Aspekt, der diese Entscheidung unterstützt, ist die Tatsache, dass der Kontroll-PC dichter am Wettkampfgeschehen platziert wird als der Auswerter-PC. Entsprechend sind Entscheidungen, wann ggf. ein Lauf manuell beendet wird, dort wesentlich besser zu treffen.

Für die Übertragung der Daten vom Kontroll- zum Auswerterrechner wurde gemeinsam eine Java-Klasse mit dem Namen “MQTT_Evaluate” erstellt. Die Klasse ist lediglich ein Datengefäß und enthält bis auf Getter- und Setter-Methoden keinerlei Anwendungslogik. Beide Anwendungen nutzen außerdem die Google-Bibliothek GSON, um MQTT-Evaluate-Objekte in Json-Strings umzuwandeln. So kann das Objekt einfach als Zeichenkette per MQTT verschickt werden. Dies hat zwar einen leichten Geschwindigkeitsnachteil gegenüber einem direkten Verschicken des Objekts, bietet aber deutlich einfachere Möglichkeiten zum Debuggen.

Zuletzt haben sich beide Gruppen auf ein MQTT-Topic abgestimmt. Auf diesem Topic sendet der Kontrollrechner und der Auswerter empfängt ausschließlich. Sobald der Kontrollrechner ein Objekt auf dem Topic veröffentlicht, empfängt der Auswerterrechner dieses und versucht es in ein Format umzuwandeln, welches eine Auswertung ermöglicht. Gelingt dies, kann mit der Auswertung begonnen werden.

6 Entwurf und Implementierung

Im Folgenden wird auf die einzelnen Komponenten des Systems und deren Kommunikation untereinander eingegangen. Im Abschnitt 6.1 Netzwerk und Kommunikation wird einmal der Netzwerkaufbau und die Kommunikation erläutert. Danach folgen im Abschnitt 6.2 das Zeitmesssystem, 6.3 die Zielrichter-App, 6.4 die Datenbank, 6.5 der Lenex-Austausch mit dem Programm EasyWK, 6.7 das Kontrollprogramm und 6.8 das Auswertungsprogramm.

6.1 Netzwerk und Kommunikation

In diesem Abschnitt wird erläutert, wie das Netzwerk aufgebaut ist und wie zwischen den einzelnen Komponenten kommuniziert wird.

6.1.1 Netzwerkaufbau

Der schlussendliche Netzwerkaufbau sieht wie folgt aus:

- Raspberry Pi 3 B+
  - Adresse: 192.168.213.1
  - Netzmaske: 255.255.255.0
- 6x Raspberry Pi Zero W
  - Adressen: 192.168.213.32 – 192.168.213.37
  - Netzmaske: 255.255.255.0
  - Gateway: 192.168.213.1
- 5x Smartphone
  - Konfiguration über DHCP
- Kontrollrechner
  - Adresse: 192.168.213.66
  - Netzmaske: 255.255.255.0
  - Gateway: 192.168.213.1
- Auswertungsrechner
  - Konfiguration über DHCP

Alle Geräte verbinden sich drahtlos mit dem Raspberry Pi 3, welcher den Netzwerkverkehr routet und einen DHCP Server bereitstellt.

Da das gesamte System in einem Schwimmbad benutzt werden soll, ist eine WLAN Verbindung hier von großem Vorteil, da es den Aufbau vor dem Wettkampf stark vereinfacht und andernfalls benötigte LAN Kabel “Stolperfallen” wären.

6.1.2 Kommunikation zwischen den einzelnen Komponenten

Für die Kommunikation zwischen den Teilkomponenten wird MQTT[1] verwendet. Aufgrund entscheidender Vorteile beim MQTT-Protokoll hat die Gruppe sich entschieden diesen Standard für alle Komponenten im System zu nutzen. Es befindet sich in Layer 7 des OSI-Schichtenmodells, das Protokoll und der MQTT-Server sichern das Weiterleiten der Nachrichten und das Ankommen der Nachrichten über QoS ab. Einer der Hauptvorteile ist die n zu m Verteilung der Nachrichten, hierfür müssen Topics definiert werden. Die Clients können auf Topics publishen und somit Nachrichten für andere Teilnehmer zu Verfügung stellen. Aktualisierungen von Topics erhalten die Clients nur wenn sie sich auf das Topic *subscribed* haben. Ein Beispiel dafür und die genauere Erklärung befindet sich in der Grafik im Kapitel 5.1.2 MQTT.

In diesem Kapitel werden alle definierten MQTT-Topics und deren Aktoren, die auf die Topics schreiben oder lesen, beschrieben. Die Daten werden in den Topics als String hinterlegt. Für jedes Topic ist ein Format definiert, in dem die Daten dort abgelegt werden müssen.

6.1.2.1 event/referees/ (Kommunikation: Zielrichter <--> Kontrollrechner)

In diesen Topics wird die Kommunikation zwischen Kontroll-Rechner und Zielrichter App abgebildet.

event/referees/contestData

Alle Wettkämpfe und Läufe inklusive Zusatzinformationen werden in diesem Topic abgelegt.

- Format:

{ "contests" : [{ "contestID" : 1, "raceID" : 1, "finished" : true } ,{ "contestID" : 2, "raceID" : 1, "finished" : false }] }

“contests” ist eine Liste von Wettbewerb/Laufkombinationen.

Ein Contest Datensatz in einer Liste besteht aus mehreren Attributen. Das Attribut “contestID”, ist die Nummer des Wettbewerbs. “raceID” beinhaltet die Laufnummer, über das Paar aus “contestID” und “raceID” kann ein Datensatz eindeutig identifiziert werden. Das “finished” Attribut zeigt, ob ein Lauf bereits beendet wurde (Feld = true) oder nicht(Feld = false).

- Aktoren:
    - Schreibend
        - Zielrichter-App
        - Kontrollrechner
    - Lesend
        - Zielrichter-App
        - Kontrollrechner

event/referees/configs/[TOKEN]

Topic für die Konfiguration der ZR-App Benutzer. Der Platzhalter [Token] steht für den jeweiligen Token eines Benutzers. Dadurch können mehrere Konfigurationen abgelegt und unterschieden werden. Über dieses Topic wird der Login abgebildet. Dies passiert indem nur Benutzer Zugriff haben hinter dessen Token das folgende Format abgelegt ist.

- Format:

{ "role": "master", "id": 1 }

Im Feld “role” befindet sich die Rolle des Benutzers. Es gibt die Rollen “master” und “normal”. Der Master hat mehr Berechtigungen als der normale Benutzer. Es darf nur einen Master geben. Im Feld “id” befindet sich die zugeordnete ID, über den der Benutzer eindeutig identifiziert werden kann.

- Aktoren:
    - Schreibend
        - Kontrollrechner
    - Lesend
        - Zielrichter-App

event/referees/raceData

Der aktuell zu bearbeitende Lauf für die Zielrichter wird in diesem Topic abgelegt, dies geschieht durch den Kontrollrechner. Die Zielrichter-App kann über dieses Topic einen neuen Lauf anfordern.

- Format:
    - Vom Kontrollrechner abgelegt

{ "contestID" : 1, "contestName" : "100 m Brust weiblich", "raceID" : 1, "lines" : [2,3,4] }

Im Attribut “contestID” steht die Wettkampfs-Nummer. Zusammen mit “raceID”, worin die Laufnummer steht, wird der Lauf eindeutig identifiziert. Über “contestName” ist angegeben wie die Bezeichnung des Wettkampfs lautet. In der Liste “lines” sind die Bahnnummern hinterlegt, die für diesen Lauf belegt sind.

    - Wenn ein neuer Lauf von der ZR-App gefordert wird, muss folgendes Format in diesem Topic abgelegt werden.

{ "contestID" : 1, "raceID" : 1}

- Aktoren:
    - Schreibend
        - Kontrollrechner, Zielrichter-App
    - Lesend
        - Zielrichter-App

event/referees/raceResults/[TOKEN]/

In diesem Topic werden die Ergebnisse der Zielrichter hinterlegt. Jeder Zielrichter hinterlegt bei seinem [Token] die Ergebnisse. Dadurch kann es zwischen den Zielrichtern nicht zu inkonsistenten Daten kommen, da jeder Zielrichter nur auf sein eigenes Topic schreiben darf. Die Ergebnisse in diesen Topics sind sofort verfügbar, sobald die Zielrichter ihre Daten in der App abgegeben haben.

- Format:

{ "referee_id" : "123", "contest_id" : "1", "race_id" : "2",

"result" : [

{"position" : "1", "tracks" : [3,2] },

{"position" : "2", "tracks" : [] },

{"position" : "3", "tracks" : [1] }

]

}

Im Feld “referee_id” wird die ID des Zielrichters hinterlegt. “contest_id” beinhaltet die Wettkampfnummer und “race_id” die Laufnummer. In der Liste “result” ist das Ergebnis vom Zielrichter hinterlegt. In dieser Liste gibt es pro Bahn einen Datensatz, in oben gezeigtem Beispiel gehen wir von 3 Bahnen aus. Jeder Datensatz enthält die Position, in der ersichtlich wird welche Platzierung der Schwimmer hat, im Feld “position”. Im Array “tracks” sind die Bahnen hinterlegt die, der Zielrichter auf den jeweiligen Positonen gesehen hat. In unserem Beispiel wurden Bahn 3 und 2 auf Platz 1 gesehen. Die Bahn 1 wurde auf Platz 3 gesehen.

- Aktoren:
    - Schreibend

        - Zielrichter-App
    - Lesend
        - Kontrollrechner

event/referees/resultOverview

Durch dieses Topic wird ersichtlich welche Zielrichter ihr Ergebnis zum aktuellen Lauf schon abgegeben haben und welcher nicht.

- Format:

{ “contestID”: 1,”raceID”: 1,

“results” : [

{ "referee" : 1, "finished" : true },

{ "referee" : 2, "finished" : false },

{ "referee" 3, "finished" : true }] }

Die Attribute “contestID” und “raceID” bilden die Wettkampfs- bzw. die Laufnummer ab. Das Attribut “results” bildet eine Liste mit den Abgabeinformationen zu den Zielrichtern ab. Ein Datensatz der Liste besteht aus “referee” und “finished” Feldern, in dem die Zielrichter ID und das abgegeben (true) oder nicht abgegeben (false) Kennzeichen hinterlegt sind.

- Aktoren:
    - Schreibend
        - Kontrollrechner
    - Lesend
        - Zielrichter-App

event/referees/forceLogout/[TOKEN]

Hierüber kann der Kontrollrechner die Apps zwingen sich auszuloggen. Dafür muss in dem jeweiligen Topic für die Benutzer ein true hinterlegt werden. Die Zielrichter werden über den Token identifiziert. Sobald der Zielrichter sich zwangsweise ausgeloggt hat, schreibt dieser selbst wieder ""(Leerstring) in das Topic. Dadurch wird verhindert, dass der Benutzer beim erneutem anmelden direkt wieder abgemeldet wird.

- Format:
    - Wenn der Kontrollrechner einen Benutzer zwingen will sich auszuloggen true
    - Andernfalls “” (Leerstring)
- Aktoren:
    - Schreibend
        - Kontrollrechner, Zielrichter-App
    - Lesend
        - Zielrichter-App

event/referees/loggedOut/[TOKEN]

Wenn ein Benutzer sich aus der Zielrichter-App ausloggt, schreibt dieser hier true rein. Dadurch bekommt der Kontrollrechner mit, dass sich ein Benutzer ausgeloggt hat. Wenn der Kontrollrechner diese Änderung wahrgenommen hat, schreib dieser anschließend einen “”(Leerstring) in das Topic.

- Format:

    - Sobald sich ein Benutzer ausgeloggt hat true
    - Andernfalls “”(Leerstring)
- Aktoren:
    - Schreibend
        - Zielrichter-App
    - Lesend
        - Kontrollrechner

event/referees/status/[TOKEN]

Auf dieses Topic muss die Zielrichter-App alle 5 Sekunden etwas publishen. Dadurch wird dem Kontrollrechner ein Ping gesendet und dieser weiß, dass die App noch verbunden ist.

- Format:
    - beliebig
- Aktoren:
    - Schreibend
        - Zielrichter-App
    - Lesend
        - Kontrollrechner

event/referees/status/statusOverview

In diesem Topic ist der Onlinestatus der Zielrichter hinterlegt.

- Format:

{ "statusOverview" : [

{"refereeId" : 1,"state": "online"},

{"refereeId" : 2,"state": "offline"},

{"refereeId" : 3,"state": "online"}]}

In der Liste “statusOverview” sind die Datensätze aller Zielrichter hinterlegt. Im Feld “refereeId” steht die ID des Zielrichters. Im Attribut “state” ist hinterlegt, ob der Zielrichter online oder offline ist.

- Aktoren:
    - Schreibend
        - Kontrollrechner
    - Lesend
        - Zielrichter-App

6.1.2.2 event/timer/ (Kommunikation: Zeitmesssystem --> Kontrollrechner)

Auf die Topics „*/time“ werden die Zeitstempel geschrieben, die die Zeit in Millisekunden seit dem 1.1.1970 angeben.

event/timer/start/time

- Beschreibung: Zeitstempel vom Start
- Aktoren:

- Schreibend
    - Raspberry Pi, welcher das Startsignal abgreift
- Lesend
    - Kontrollrechner

event/timer/[BAHN.SEITE]/time

- Beschreibung: Zeitstempel der einzelnen Bahnen; sowohl Zwischen- als auch Endzeiten
- Aktoren:
    - Schreibend
        - Raspberry Pi der jeweiligen Bahn
    - Lesend
        - Kontrollrechner

Auf die Topics „*/status" wird alle 10 Sekunden eine Meldung geschrieben, die signalisieren soll, dass sich der entsprechende Pi im Netzwerk befindet und Nachrichten schicken kann. Kommt dieses Signal nicht an, ist davon auszugehen, dass zurzeit ebenfalls keine Zeitstempel verschickt werden können.

event/timer/start/status

- Beschreibung: Statussignal des Raspberry Pis, der das Startsignal erfasst
- Aktoren:
    - Schreibend
        - Raspberry Pi, welcher das Startsignal abgreift
    - Lesend
        - Kontrollrechner

event/timer/[BAHN.SEITE]/status

- Beschreibung: Statussignal der Raspberry Pis auf den einzelnen Bahnen
- Aktoren:
    - Schreibend
        - Raspberry Pi der jeweiligen Bahn
    - Lesend
        - Kontrollrechner

event/evaluation (Kommunikation: Kontrollrechner --> Auswerter-PC)

event/evaluation/

Dieses Topic enthält alle wettkampfbezogenen Daten des aktuellen Wettkampfes.

- Format:

{"contestID":3,"raceID":1,"heatID":700001,"contestName":"100m Schmetterling weiblich", "timeData":{"1":[],"2":[27546],"3":[30681],"4":[33249],"5":[]},"ageData":{"1":2006,"2":2005,"3":200 6,"4":2004,"5":2007},"refereeData":["{\"result\":[{\"position\":1,\"tracks\":[2]},{\"position\":2,\"trac

ks\":[3]},{\"position\":3,\"tracks\":[4]},{\"position\":4,\"tracks\":[]},{\"position\":5,\"tracks\":[]},{\"position\":6,\"tracks\":[]},{\"position\":7,\"tracks\":[]},{\"position\":8,\"tracks\":[]}],\"referee_id\":\"4\"}","{\"result\":[{\"position\":1,\"tracks\":[2]},{\"position\":2,\"tracks\":[3]},{\"position\":3,\"tracks\":[4]},{\"position\":4,\"tracks\":[]},{\"position\":5,\"tracks\":[]},{\"position\":6,\"tracks\":[]},{\"position\":7,\"tracks\":[]},{\"position\":8,\"tracks\":[]}],\"referee_id\":\"1\"}","{\"result\":[{\"position\":1,\"tracks\":[2]},{\"position\":2,\"tracks\":[3]},{\"position\":3,\"tracks\":[4]},{\"position\":4,\"tracks\":[]},{\"position\":5,\"tracks\":[]},{\"position\":6,\"tracks\":[]},{\"position\":7,\"tracks\":[]},{\"position\":8,\"tracks\":[]}],\"referee_id\":\"3\"}","{\"result\":[{\"position\":1,\"tracks\":[2,3]},{\"position\":2,\"tracks\":[]},{\"position\":3,\"tracks\":[4]},{\"position\":4,\"tracks\":[]},{\"position\":5,\"tracks\":[]},{\"position\":6,\"tracks\":[]},{\"position\":7,\"tracks\":[]},{\"position\":8,\"tracks\":[]}],\"referee_id\":\"5\"}","{\"result\":[{\"position\":1,\"tracks\":[2]},{\"position\":2,\"tracks\":[3]},{\"position\":3,\"tracks\":[4]},{\"position\":4,\"tracks\":[]},{\"position\":5,\"tracks\":[]},{\"position\":6,\"tracks\":[]},{\"position\":7,\"tracks\":[]},{\"position\":8,\"tracks\":[]}],\"referee_id\":\"2\"}"]}

- Aktoren:
  - Schreibend
    - Kontrollrechner
  - Lesend
    - Auswerter-PC

6.2 Zeitmesssystem

Die Aufgabe des Zeitmessprogramms ist es, bei eingehenden Signalen von den Drückern an den GPIO (General Purpose Input Output) der Raspberry Pis einen Zeitstempel zu erstellen und über MQTT an den Kontrollrechner zu senden, außerdem werden die Zeitstempel in einer Backupdatei auf dem Raspberry Pi gespeichert.

Das Programm wurde in der Programmiersprache C mit dem Konsoleneditor Nano entwickelt und besteht aus folgenden Source-Dateien: *main.c*, *mqtt.c*, *backup.c*, *stopwatch.c, test.c* und den dazu gehörigen Header-dateien. Das Kompilieren des Programms findet über ein *Makefile* statt und wird mit dem Befehl *make clean* ausgeführt. Im Folgenden wird auf die einzelnen Funktionen des Programmes genauer eingegangen.

6.2.1 Programm

6.2.1.1 MQTT-Schnittstelle

Genutzte Library

Wir benutzen die Paho MQTT Client Library für C. Der Client wird bei uns im asynchronen Modus erstellt. Das bedeutet, dass die Library automatisch im Hintergrund mehrere Threads, z.B. für Handshakes und zum Aufrechterhalten der Verbindung, erstellt.

Die Quality of Service(QOS) ist bei uns auf 1 gestellt, was bedeutet, dass versendete Nachrichten mindestens einmal ankommen. Eine QOS von 0 würde bedeuten, dass die Nachricht immer genau einmal verschickt wird; ein Ankommen der Nachricht wäre somit nicht gewährleistet. Eine QOS von 2 garantiert, dass die Nachricht genau einmal ankommt.

mqtt.c/mqtt.h

In dieser Datei befinden sich alle Methoden, die für die MQTT-Kommunikation benötigt werden. Es kann ein Client auf dem Gerät instanziiert werden, die Verbindung zum Broker hergestellt werden oder Nachrichten auf einen Topic gepublished werden.

Es gibt sogenannte Callback-Funktionen, die, wie bereits erwähnt, automatisch zu bestimmten Zeitpunkten, z.B. wenn die Verbindung zum Broker verloren geht, aufgerufen werden. Dieses sind bei uns *delivered()*, *msgarrvd()* und *connlost()*. Diese können vom Programmierer frei benannt und implementiert werden und werden über die *MQTTClient_setCallbacks()*-Methode angemeldet.

msgarrvd() würde in der Praxis primär dafür verwendet, um sich auf ein Topic zu subscriben. Auch wenn wir keine Subscription durchführen, benötigen wir diese Funktion, um das Acknowledgment für eine versendete Nachricht zu erhalten.

Funktionen die von uns aktiv im Code aufgerufen werden können heißen *instantiateClient()*, *connectToBroker()*, *sendMessage()* und *disconnectFromBroker()*.

Bei der Instanziierung des Clients und beim Herstellen der Verbindung ist es wichtig, dass dies ausgeführt wird, damit bei einem Scheitern kein menschliches Eingreifen nötig ist. Wenn es beim ersten Mal nicht funktioniert, wird es immer wieder versucht.

Als Beispiel folgt die *connectToBroker()*-Methode:

```
void connectToBroker(void){
        int rc;
        MQTTClient_connectOptions conn_opts =
                        MQTTClient_connectOptions_initializer;
        conn_opts.keepAliveInterval = 20;
        conn_opts.cleansession = 0;
        do{
                //printf("Trying to connect\n");
                rc = MQTTClient_connect(client, &conn_opts);
        } while(rc != MQTTCLIENT_SUCCESS);
        //printf("Connected to Broker");
}
```

Abbildung 10 connectToBroker Funktion

Geht die Verbindung im laufenden Betrieb verloren, ist in der *connlost()*-Methode definiert, dass die *connectToBroker()* Funktion aufgerufen wird.

6.2.1.2 Entprellung

Was ist Prellen überhaupt?

Ein prellender Knopf wechselt nicht sauber zwischen der logischen 0 und 1 beim Drücken und Loslassen des Knopfes. Stattdessen treten schnell nacheinander folgende Nullen und Einsen auf, bis das abzugreifende Signal dann nach kurzer Zeit stabil anliegt.

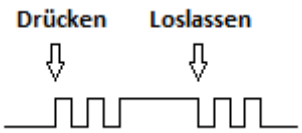


Abbildung 11 Beispiel Prellen

Wie lässt sich dieses Problem umgehen?

Einerseits kann ein Knopf hardwareseitig entprellt werden, sodass erst gar keine fehlerhaften Signale anliegen. Diese Variante erfordert jedoch spezielle Hardware und ist meistens teurer. Stattdessen kann

man dieses ungewollt auftretende Signalprellen auch softwareseitig abfangen. Letzteres wurde von uns angewandt.

Unser jetziger Entprellalgorithmus sieht wie folgt aus:

```
int buttonPressed = 0;
while(1){
        if(digitalRead(5) == 1){
                buttonPressed++;
        }else{
                buttonPressed = 0;
        }
        if(buttonPressed == DEBOUNCETIME){
                getTime(&payload);
                sendMessage((char*)TOPIC, payload);
                writeBackup(payload);
                sleep(5);
        }
}
```

Abbildung 12 Entprellen durch Mehrfachabfrage

Jedes Mal, wenn eine logische 1 an dem Pin des Knopfes anliegt, wird eine Variable hochgezählt. Liegt keine 1 mehr an, wird die Variable zurückgesetzt.

Ebenfalls wird bei jedem Durchlauf überprüft, ob die Variable mit einem vordefinierten Wert übereinstimmt. Stimmen die beiden überein wissen wir, dass eine stabile 1 anliegt und der Knopf tatsächlich gedrückt wurde. Beim mechanischen Prellen liegen die Einsen nicht lange genug an, um den Wert zu erreichen.

Nachdem eine stabile 1 erkannt wurde, schläft das Programm für 5 Sekunden, da in unserem Anwendungsfall niemals zwei Zeiten in solch kurzer Abfolge gestoppt werden können. Dies verhindert das Prellen beim Loslassen des Knopfes. Der vordefinierte Wert „DEBOUNCETIME“ gibt hier die Anzahl der Schleifendurchläufe der Endlosschleife an.

Dieser Algorithmus ist jedoch sowohl von der Geschwindigkeit des Prozessors und dem Betriebssystem abhängig als auch von dem Programm und dem verwendeten Drücker. Ebenfalls ist es nicht möglich den benötigten Wert für DEBOUNCETIME auszurechnen, sondern dieser muss durch Ausprobieren herausgefunden werden. Schlussendlich haben wir DEBOUNCETIME=7500 gewählt.

Dieser Algorithmus funktioniert jedoch mit unserem jetzigen Aufbau und wurde gründlich getestet.

Ein anderer Algorithmus würde wie folgt aussehen:

```
int buttonPressed = 0;
while(1){
        if(digitalRead(5) == 1){
                buttonPressed++;
        }else{
                buttonPressed = 0;
        }
        if(buttonPressed == DEBOUNCETIME){
                getTime(&payload);
                sendMessage((char*)TOPIC, payload);
                writeBackup(payload);
        }
        msleep(5);
}
```

Abbildung 13 Entprellen durch Mehrfachabfrage mit Timer

Bei diesem Algorithmus gibt es erneut eine Variable, die zum Zählen der Schleifendurchläufe verwendet wird, jedoch läuft das Programm auf einem globalen Timer. Die Schleife wird alle 5 Millisekunden einmal ausgeführt. Wenn man nun also den Wert DEBOUNCETIME auf 3 setzt bedeutet das, dass ein stabiles Signal anliegt, wenn der Knopf für 15 Millisekunden gedrückt wurde.

Somit wäre der Wert nur noch von der Hardware des Drückers abhängig. Der Nachteil liegt jedoch darin, dass die Reaktionszeit 15-20 Millisekunden beträgt und somit eine Ungenauigkeit von bis zu 5 Millisekunden auftreten kann.

Die Anzahl der Millisekunden die geschlafen wird und die DEBOUNCETIME können jedoch auch angepasst werden, um die maximale Ungenauigkeit zu verringern. Mit einer Sleep-Time von 1 Millisekunde und einer DEBOUNCETIME von 15 wäre das Problem nur noch sehr gering und für unseren Anwendungsfall wahrscheinlich irrelevant, da sämtliche Zeiten auf Hundertstel-Sekunden abgeschnitten werden.

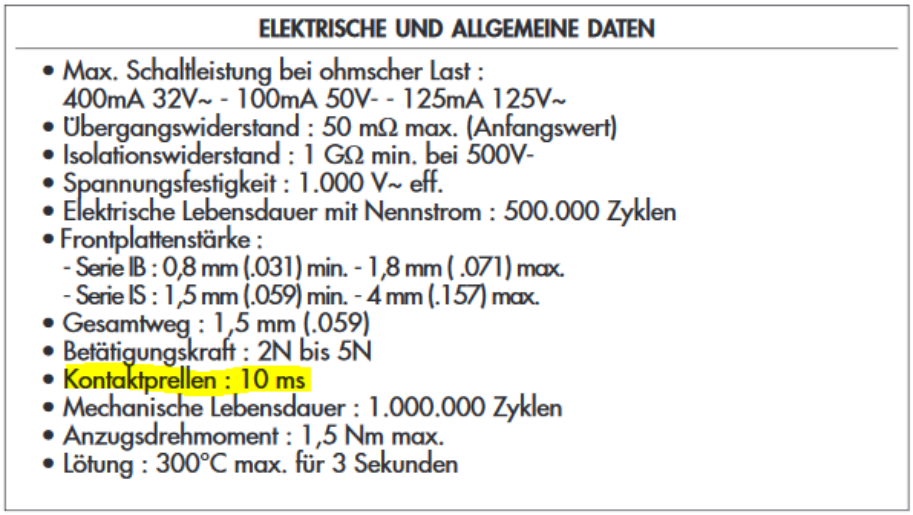
Eine letzte Alternative würde wie folgt aussehen:

```
while(1){
        if(digitalRead(5) == 1){
                msleep(DEBOUNCETIME);
                getTime(&payload);
                sendMessage((char*)TOPIC, payload);
                writeBackup(payload);
                while(digitalRead(5) == 1){}
                msleep(DEBOUNCETIME);
        }
}
```

Abbildung 14 Überbrückung des Prellens durch Verzögerung

In diesem Beispiel wird solange geprüft, bis eine 1 am Pin anliegt. Danach wird das Prellen mit einem Sleep überbrückt. Jetzt wird so lange gewartet, wie die 1 weiterhin anliegt und danach wird das Prellen beim Loslassen des Knopfes erneut mit einem Sleep überbrückt.

Diese Variante ist ebenfalls nur von der Hardware des Drückers abhängig. In unserem Fall wären dies 10 Millisekunden, die der Drücker maximal prellt. Zu finden ist dieser Wert ist den Hardwarespezifikationen des jeweiligen Knopfes.



ELEKTRISCHE UND ALLGEMEINE DATEN

- Max. Schaltleistung bei ohmscher Last :
  400mA 32V~ - 100mA 50V- - 125mA 125V~
- Übergangswiderstand : 50 mΩ max. (Anfangswert)
- Isolationswiderstand : 1 GΩ min. bei 500V-
- Spannungsfestigkeit : 1.000 V~ eff.
- Elektrische Lebensdauer mit Nennstrom : 500.000 Zyklen
- Frontplattenstärke :
  - Serie IB : 0,8 mm (.031) min. - 1,8 mm (.071) max.
  - Serie IS : 1,5 mm (.059) min. - 4 mm (.157) max.
- Gesamtweg : 1,5 mm (.059)
- Betätigungskraft : 2N bis 5N
- Kontaktprellen : 10 ms
- Mechanische Lebensdauer : 1.000.000 Zyklen
- Anzugsdrehmoment : 1,5 Nm max.
- Lötung : 300°C max. für 3 Sekunden

Abbildung 15 Allgemeine Daten Knopf

6.2.1.3 Backups

Aufbau des Backups

Laut Pflichtenheft war es gefordert, dass von jedem erzeugten Zeitstempel auch ein Backup anzulegen ist, welches im CSV-Format lokal auf dem Raspberry Pi gespeichert wird. CSV steht für *comma-separated values*. Ziel dieses Backups ist, dass auch bei Übertragungsproblemen zum Kontrollrechner immer eine Sicherheitskopie existiert, mit der man manuell Zeiten rekonstruieren kann.

Wir haben uns dafür entschieden, in das Backup sowohl die Zeit im menschlich-lesbaren Format als auch den eigentlichen Zeitstempel der verschickt wird, zu schreiben.

Ein beispielhaftes Backup würde wie folgt aussehen:

```
date and daytime, timestamp
14.12.2018 15:22:38, 1544797358165
14.12.2018 15:22:56, 1544797376738
```

backup.c/backup.h

Von uns wurden die Funktionen *backupInit()*, welche einmalig beim Start aufgerufen wird, sowie *writeBackup()*, welche bei jedem verschickten Zeitstempel aufgerufen wird, implementiert.

Im *backupInit()* wird der Name der Datei, welcher dem Datum in Form von „DD.MM“ entspricht, festgelegt und in einer globalen Variable gespeichert. Ebenfalls werden die Überschriften der zwei Spalten in die Datei geschrieben.

In der *writeBackup()*-Funktion werden dann die eigentlichen Zeitstempel an die Datei angehängt.

Sollte es zu einem zwischenzeitigen Ausfall des Gerätes im laufenden Schwimmwettkampf kommen, wird an die bereits erstellte Datei erneut die Überschriften-Zeile angehängt.

Die Backups sind im Ordner */home/pi/* zu finden.

6.2.1.4 Zeitstempel

Format

Die von den Drückern generierten Zeiten geben die Millisekunden, die seit 00:00:00 am 1. Januar 1970 vergangen sind, an.

stopwatch.c/stopwatch.h

Die wichtigste Funktion hier ist *getTime()*. Diese erzeugt die vorher erwähnten Zeitstempel.

Ebenfalls wurden die Funktionen *getDaytime()* und *getDate()* implementiert. *getDaytime()* gibt die UNIX-Zeit in menschlich-lesbarer Form zurück und *getDate()* gibt das jetzige Datum mit Tag und Monat zurück. Letztere beide sind lediglich nötig, um die Lesbarkeit und Zuordnung der Backups zu vereinfachen.

6.2.1.5 WiringPi

WiringPi ist eine in C geschriebene Bibliothek zur Programmierung der GPIO des Raspberry Pis, welche benötigt wurde, um die Signale des Drückers[GH1] zu empfangen und weiterzuverarbeiten. Die Ansteuerung erfolgt über die Pins 17 (3V Power) und 18 (GPIO 24) und wird[GH2] mit den Funktionen wiringPiSetup() und pinMode(5, INPUT) initialisiert. Der Pin 18 wird in einer Endlosschleife über digitalRead(5) abgefragt und gibt bei anliegender Spannung eine Eins zurück.

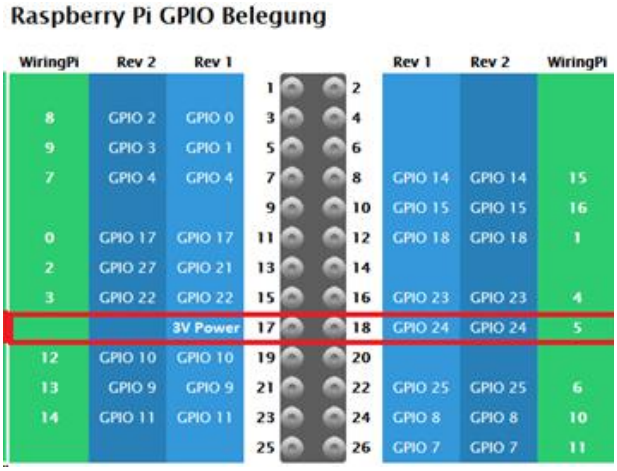


Abbildung 16 GPIO Belegung

6.2.2 Komponenten unseres Systems

6.2.2.1 Raspberry Pi Zero W

Wir haben uns für Raspberry Pi entschieden, da es die verbreitetste Marke ist und somit eine Vielzahl an Libraries und Dokumentationen im Internet bereitstehen. Zudem bieten die Geräte durch das eigene Betriebssystem „Raspbian“, welches auf Debian basiert, viele Konfigurationsmöglichkeiten für alle möglichen Einsatzgebiete.

Abbildung 17 Raspberry Pi Zero W (Abbildung aus [2])

Für den Zero W speziell haben wir uns entschieden, da er mit Abstand das kleinste Modell ist. Mit einer Länge von 65mm und einer Breite von 30mm ist er kleiner als die meisten Stoppuhren.

Ebenfalls hat er einen geringeren Stromverbrauch als die anderen Modelle und ist preislich günstiger. Der langsamere Prozessor spielt für uns keine Rolle, da unser Programm für unseren Anwendungsfall nicht viel Rechenleistung benötigt.

Ein weiterer Vorteil des Zero W ist, dass dieser bereits über einen eingebauten WLAN-Chip verfügt und somit kein extra USB-Adapter erforderlich ist, um diese Funktionalität bereitzustellen.

Der größte Nachteil für unseren Anwendungsfall liegt darin, dass die Raspberry Pis keine Real-Time-Clock bieten. Somit kann die Systemuhr schnell zu ungenau werden, um verlässliche Zeiten zu liefern.

6.2.2.2 Raspberry Pi 3 Model B+

Abbildung 18 Raspberry Pi 3 B+ (Abbildung aus [3])

Für den Access-Point fiel die Entscheidung auf ein Raspberry Pi 3 Model B+, der einen CYW43455 WLAN-Chip besitzt, die Dual-Band (2,4GHz und 5GHz) Übertragung unterstützt und eine Geschwindigkeit von 100Mbit/s erreichen kann. Bei der Konfiguration sollte darauf geachtet werden, dass das Power-Management ausgeschaltet ist, da der Access Point sonst bei nicht Benutzung (< 10 Minuten) in einen Sleep-Mode/Power-Save-Mode übergeht. Dies führt dazu, dass die WLAN-Verbindungen unterbrochen werden und der Access Point manuell wieder „aufgeweckt“ werden muss. Der gesamte Netzwerkverkehr läuft über dieses Gerät, daher hat man hier ein leistungsstärkeres Modell der Raspberry Pi Familie gewählt.

Zusätzlich wird ein USB-WLAN-Adapter benutzt. Dieser hat sich in unseren Tests als zuverlässiger, mit einer höheren Sendereichweite, herausgestellt als der Onboard-WLAN-Adapter. Bei der Auswahl des Adapters ist darauf zu achten, dass er den AP(Access-Point)-Modus unterstützt.

Da die Reichweite des Access Point bei freier Fläche ungefähr 20 Meter beträgt und dies bei Schwimmwettkämpfen mit 50 Meter Bahnen oder schlechten Räumlichkeiten zu Problemen führen kann, sollte für weitere Projekte eine zusätzliche Antenne verbaut werden. Des Weiteren gäbe es die Möglichkeit den Raspberry Pi durch einen leistungsstarken NTP/PTP-fähigen Router zu ersetzen.

6.2.2.3 Drücker

Die Drücker bestehen aus Polyoxymethylen. Sie sind 12cm lang und besitzt einen Durchmesser von 25mm. Im Inneren befindet sich ein Knopf und ein eingelöteter Kurzschlusswiderstand von 18 Ohm. Anzuschließen sind sie mithilfe eines Mikrofon-Einbausteckers. (vgl. [4])

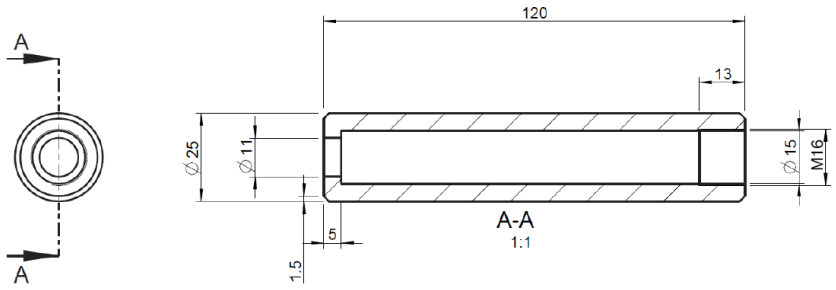


Abbildung 19 Konstruktionszeichnung aus Projektdokumentation 2018 (Abbildung aus: [4])

Die Drücker wurden vollständig von der vorherigen Projektgruppe übernommen; von uns wurden lediglich die Kabel auf eine Länge von knapp zwei Meter gekürzt.

6.2.2.4 Gehäuse

Im Laufe des Projekts kam es zum Einsatz von zwei verschiedenen Gehäusen für die Raspberry Pis, da die finalen Gehäuse aufgrund von einer langen Lieferzeit (> 1 Monat) nicht für die Testläufe und das Adventsschwimmen genutzt werden konnten, musste übergangsweise ein alternatives Gehäuse eingesetzt werden.

Für die finalen Gehäuse gab es einige Voraussetzungen. Die wichtigste war, dass diese absolut wasserdicht sind, da sie sich bei den Schwimmwettkämpfen immer in der Nähe des Schwimmbeckens befinden und die Gefahr besteht, dass diese versehentlich ins Wasser fallen oder durch Spritzwasser die Elektronik im Gehäuse beschädigt wird. Des Weiteren sollten die Gehäuse ein Fenster besitzen, damit spätere Erweiterungen wie beispielsweise LEDs zur Statusanzeige oder ein kleines LC-Display möglich sind. Aus diesen Gründen wurde sich für das auf der Abbildung 21 zu sehende Gehäuse entschieden, welches die Schutzklasse IP67 hat und somit die Elektronik gegen feste Fremdkörper mit einem Durchmesser <= 1,0 mm und zeitweiliges Untertauchen im Wasser schützt. Das Gehäuse ist 3,4 cm hoch, 5,7 cm breit, 8,2 cm lang und wurde um einen 4-poligen Stecker an der Seite für den Drücker erweitert.

Abbildung 21 finale Gehäuse

Abbildung 20 vorrübergehende Gehäuse

6.2.2.5 Stromversorgung Raspberry Pi Zero

Da die Raspberry Pis über keine eigene mobile Stromversorgung verfügen, diese in Schwimmbädern aus Sicherheitsgründen nicht mit Standardnetzteil geladen werden können und nicht immer genügend Steckdosen vorhanden sind, kam hier das Zero LiPo und ein Lithium-Ionen-Akku zum Einsatz.

Das Zero LiPo ist eine mobile Netzteilplatine für alle Raspberry Pi Versionen, welche direkt auf die GPIO-Pins gelötet wurde. Über den 2-poligen JST-Anschluss kann ein passender Akku angeschlossen werden. Auf der Platine befinden sich zwei LEDs. Eine blaue LED zeigt an, dass ein geladener Akku angeschlossen ist, eine rote LED symbolisiert einen niedrigen Akkustand. Der dazu verwendete Lithium-Ionen-Akku mit 2000mAh kommt auf eine Akkulaufzeit von ca. 14 Stunden bei normaler Auslastung (Zeitmessung läuft) des Raspberry Pis.

Abbildung 22 Zero LiPo

6.2.3 Zeitsynchronisation

6.2.3.1 PTP

PTP steht für „Precision Time Protocol"([5]). Dies ist ein Protokoll, um die Uhren mehrerer Geräte in einem lokalen Netzwerk zu synchronisieren. Es wird eingesetzt, damit die Uhren aller Raspberry Pis möglichst synchron sind, um die Zeiten so genau wie möglich messen zu können.

Mit PTP über WLAN und lediglich Software-Timestamping, wie es bei uns der Fall ist, kann eine durchschnittliche Abweichung von weniger als 10 Mikrosekunden erreicht werden.(vgl. [5])

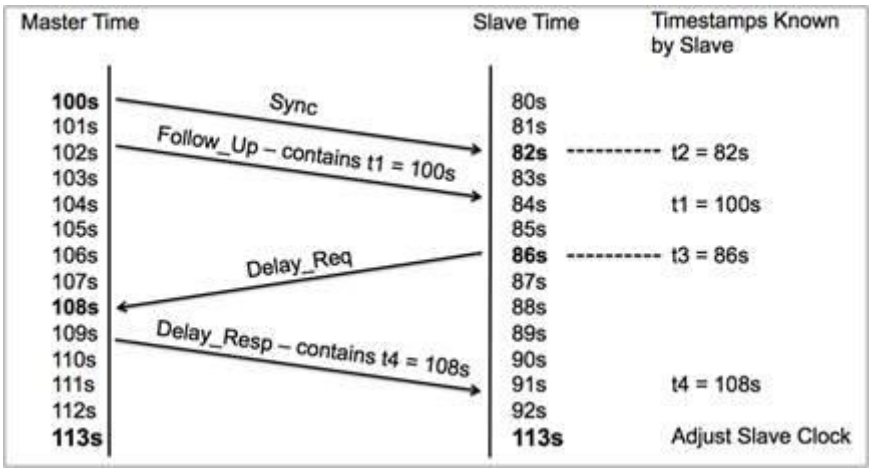


Abbildung 23 PTP Traffic (Abbildung aus: [6])

In Abbildung 23 ist die Funktionsweise von PTP dargestellt. Zuerst wird vom Master ein *Sync* an alle Slaves gesendet. Der Master merkt sich wann er das Packet versendet hat (*t1*) und alle Slaves merken sich wann sie die Pakete empfangen haben (*t2*). In der darauffolgenden Nachricht vom Master schickt er den Zeitstempel mit, zu der er das *Sync* verschickt hat; in diesem Fall also 100s. Unterstützt der verwendete Netzwerkadapter Hardware-Timestamping können diese beiden Nachrichten auch zu einer zusammengefasst werden.

Danach schickt der Slave ein *Delay_Req* an den Master. Er merkt sich den Zeitpunkt des Versendens (*t3*) und der Master den Zeitpunkt des Ankommens (*t4*). Schlussendlich schickt der Master dem jeweiligen Slave die *Delay_Resp*, in der der Zeitpunkt enthalten ist, zu dem er den *Delay_Req* erhalten hat.

Dem Slave sind nun die Zeitpunkte t1, t2, t3 und t4 bekannt. Aus diesen Werten kann er die durchschnittliche Netzwerklatenz berechnen. Mit den Zahlen aus dem Beispiel kommt man hier auf den folgenden Wert:

Latenz = ((t2 – t1) + (t4 – t3)) / 2 --> ((82 – 100) + (108 – 86)) / 2 = 2

Bei dem Algorithmus wird stets davon ausgegangen, dass der Weg vom Master zum Slave genau so lang ist, wie der Weg vom Slave zum Master.

Schlussendlich kann der Slave den Offset berechnen mit den Werten:

Offset = t2 – t1 – Latenz --> 82 – 100 – 2 = -20

Der Slave weiß also, dass er 20 Sekunden hinter dem Master liegt und kann seine Clock entsprechend anpassen.

Diese Berechnung findet zwischen dem Master und jedem Slave statt und kann mehrmals pro Sekunde passieren.

6.2.3.2 NTP

Das Network Time Protocol (NTP) ist ein Standard-Internetprotokoll, welches das verbindungslose Transportprotokoll UDP (Port 123) verwendet, um die Zeitsynchronisation in Computernetzwerken zu gewährleisten. NTP wurde als Hintergrundprozess (daemon) ntpd für alle UNIX-artigen und Windows Betriebssystemen implementiert und kann nicht nur die Zeit seiner eigenen Rechneruhr synchronisieren, sondern auch die eigene Zeit als Server für andere Systeme bereitstellen.

Mithilfe dieser Möglichkeit kann auf einfache Weise eine hierarchische Zeitverteilung in einem Netzwerk eingerichtet werden. Die einzelnen Ebenen werden Stratum genannt und ein kleinerer Stratum-Wert bedeutet dabei eine höhere Ebene in der Hierarchie-Struktur, wobei auf der obersten Ebene, Stratum 0, sich die Atomuhren befinden.

Der NTP-Daemon fragt zyklisch mehrere unabhängige Referenzzeitquellen in einem Server Pool ab (pool.ntp.org). Die einzelnen Zeitquellen werden auf den Stratum Ebenen in Gruppen zusammengefasst und anschließend gegeneinander gewichtet, um die beste Referenzzeitquelle zu selektieren.

Infolge dessen, dass die Raspberry Pis keine eigene Real Time Clock besitzen und es dadurch zu Zeitabweichung zwischen Kontrollrechner und den Raspberry Pis kommen könnte, wurde sich dazu entschieden ein NTP-Netzwerk mit dem Kontrollrechner als Zeitserver aufzubauen.

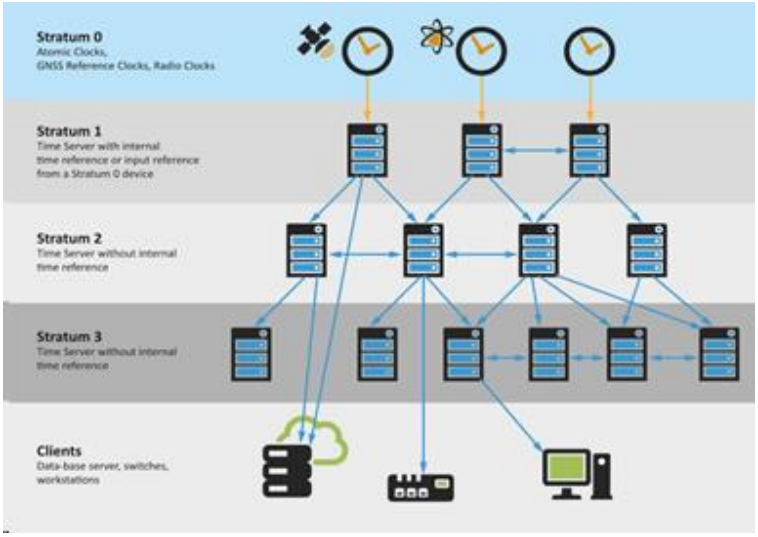


Abbildung 24 NTP Ebenen

6.2.4 Probleme

Am Anfang wurde für das Projekt ein Raspberry Pi 2 mit einem Wireless USB Adapter (EDIMAX EW-7811UN) als Access Point verwendet, welcher bei den ersten Tests mit den simulierten Raspberry Pis (Programm wurde mehrmals parallel auf einem Raspberry ausgeführt) keine Probleme aufwies. Bei späteren Tests mit den im Projekt eingesetzten Raspberry Pi Zeros, kam es nun wiederholt zu fehlerhaften Zeiten. Diese traten genau dann auf, wenn Zeitstempel bei einer unterbrochenen WLAN-Verbindung (< 10 Millisekunden) genommen wurden und der PTP-Daemon dann auf eine Reset-Time zurückgriff, die mehrere Tage in der Vergangenheit lag. Da die Unterbrechungen bei einer normalen Nutzung nicht zu erkennen waren, erwies sich die Fehlersuche als schwierig. Dennoch konnte das Problem auf eine zu hohe Auslastung des Raspberry Pi 2 zurückgeführt werden und so wurde aufgrund dessen dieser dann durch einen deutlich performanteren Raspberry Pi 3 B+ ersetzt, um ein stabiles WLAN aufzubauen.

6.3 Zielrichter-App

Wie bereits erwähnt, wird die Smartphone-App zur Abgabe der gesehenen Zieleinläufe der Zielrichter benötigt. Es soll dabei mit so wenigen Klicks wie möglich eine Abgabe erstellt werden, damit auch in schnell hintereinander folgenden Läufen eine schnelle Abgabe denkbar ist. Dabei entfällt dann auch der enorme Verbrauch von Papier und es wird eine Person weniger benötigt, welche die Zettel vorher immer zum Auswerter bringen musste. Außerdem wird die Auswertung dadurch erleichtert, dass die Zielrichterentscheide elektronisch mit den genommenen Zeiten abgeglichen werden können.

Die Smartphone App wurde innerhalb dieses Projekts für Android Smartphones entwickelt. Dafür wurden drei „Wiko Sunny 2 Plus“ und zwei „Wiko Sunny 3“ Smartphones verwendet, die von der Hochschule für dieses Projekt angeschafft wurden.

6.3.1 Recherchen

Die Einarbeitung und die damit folgenden Recherchen in die Thematik, begann für die Zielrichter-App Gruppe, wie bei allen anderen, mit einer zweiwöchigen Intensivphase. Zu Beginn musste überlegt werden in welcher Entwicklungsumgebung die App geschrieben wird. Auch musste entschieden werden, ob die App für Android und für IOS entwickelt werden sollte.

Als erstes wurde sich für Xamarin entschieden. Diese Entwicklungsumgebung dient als Cross-Plattform zum Programmieren. Da es aber Probleme mit dem Download und sehr wenig Erfahrung mit dieser IDE gab, wurde die Idee wieder verworfen.

Auf der Suche nach einer neuen Entwicklungsumgebung, kam die Gruppe auf die Idee, Visual Studio zu benutzen, da dies ein Komplettpaket für alle möglichen Anwendungsfälle wie auch die Programmierung von Handy-Apps enthält. Allerdings kam es zu dem Problem, dass die Programmiersprache C# gewesen wäre und damit niemand aus der Gruppe Erfahrung hatte. Ein weiteres Problem bestand darin, dass eine Cross-Plattform Programmierung sehr schwierig werden würde, da IOS Probleme, mit der Hauptfunktionalität Drag und Drop, gemacht hätte. Das wäre wesentlich für die spätere App gewesen. Deshalb wurde auch die Idee, der Entwicklung in Visual Studio, wieder verworfen. Nicht zuletzt auch, weil Visual Studio nur mäßig auf den PCs der Gruppenmitgliedern lief, aufgrund des gewaltigen Umfangs der IDE.

Da nun klar war, dass eine Entwicklung für Android und IOS den Rahmen sprengen würde, wurde festgelegt, dass nur noch nativ für Android entwickelt werden sollte. Die Wahl fiel auf die Entwicklungsumgebung Android-Studio. Hier würde die Programmiersprache Java verwendet werden, womit alle Gruppenmitglieder Erfahrung hatten und ein Gruppenmitglied auch schon Kenntnisse mit dieser Umgebung hatte und so sein Fachwissen sehr gut einbringen konnte.

Nachdem als Entwicklungsumgebung Android Studio festgelegt wurde, musste bezüglich der Erstellung der Tests recherchiert werden. Da das Schreiben der Tests vor der eigentlichen Implementierung stattfindet, wurde ein Framework gesucht, welches schnell zu integrieren ist und bei dem der Einarbeitungsaufwand nicht so hoch ist. Die meisten Gruppenmitglieder hatten keine Erfahrungen mit TDD in der Praxis, daher sollte es so einfach wie möglich sein die Tests zu schreiben und durchzuführen.

Um eine mobile Anwendung zu testen, gibt es verschiedene Frameworks. Innerhalb der Recherchephase wurden diese kurz geprüft, allerdings fiel die Wahl relativ schnell aus das Framework Espresso.

Einige Frameworks wie z.B. Appium waren sowohl für Android als auch iOS geeignet. Da dies eventuell den Umfang und Einarbeitungsaufwand erhöhen würde, wurden diese Frameworks nicht berücksichtigt. Außerdem waren viele Frameworks Open Source wie z.B. Robotium. Dies ist generell kein Nachteil, allerdings wurde ein Framework gesucht, welches genügend Tutorials, Einarbeitungsmaterial und eine große Community besitzt. Daher wurde beschlossen Espresso zu verwenden.

Espresso ist, wie Android, ebenfalls von Google und bereits in die Entwicklungsumgebung integriert. Daher entfiel weiterer Installations- und Einrichtungsaufwand. Außerdem stehen für Espresso einige Tutorials zur Verfügung und auf den offiziellen Androidseiten finden sich auch viele Informationen. Außerdem schien die Nutzung unkompliziert zu sein. Da es sich dabei um ein reines Android-Framework handelt, entfiel ebenfalls unnötiger Overhead und Espresso ist eine schlanke Software.

6.3.2 Anforderungen an die App

Die Anforderungen an die Zielrichter-App beliefen sich auf folgende Punkte:

Die gesamte Smartphone-App musste Android kompatibel sein. Zudem musste ein Sicherheitsfeature eingebaut werden, damit sich keine unbefugten Personen in die App einloggen konnten. Dieses bestand dann darin, dass man sich in der App mit einem 6-stelligen Code anmelden muss. Dieser Code wird von der Kontrollrechner-App generiert und innerhalb der App gespeichert, sodass man ihn nicht neu eingeben muss, wenn er sich nicht geändert hat.

Damit die Eingabe der Zielreihenfolge der Bahnen von den Zielrichtern möglichst benutzerfreundlich ist, sollte ein System mithilfe des Drag and Drop entwickelt werden. Die Eingabe sollte nicht länger als 10 Sekunden pro Anwender dauern. In diesem Abgabescreen sollten den Zielrichtern auch der aktuelle Wettkampf und die aktuelle Laufnummer angezeigt werden.

Diese Eingaben sollten über einen Bestätigungsscreen überprüft werden können und über einen weiteren Screen sollte zu sehen sein, welcher Zielrichter schon abgegeben hat und welcher noch nicht. Zusätzlich sollte es möglich sein, eine Übersicht aufzurufen, in der der Zielrichterstatus (Online oder Offline) angezeigt wird. Die App sollte auch einen weiteren Modus besitzen, in dem ein „Master“ weitere Funktionen wie „Lauf beenden“ und „Wettkampf/Lauf ändern“ zur Verfügung hat. Die Funktion “Lauf beenden” sollte mithilfe eines Buttons im Abgabenübersichtscreen realisiert werden. Den “Wettkampf ändern” über eine weitere Activity, in der der Master den Wettkampf und die Laufnummer einstellen kann.

6.3.3 Entwicklungsumgebung

Für die Entwicklung der Smartphone-App, wurde die Entwicklungsumgebung "*Android-Studio"* mit der Version 3.1.4 verwendet. Gründe hierfür waren die allen Gruppenmitgliedern bekannte Programmiersprache "*Java"* und die Tatsache, dass Android Studio einen eingebauten Editor besaß, mit dem man sehr effizient Layouts für die Activitys bauen konnte. Diese IDE lief am performantesten auf allen PCs der Gruppenmitglieder. Der integrierte Emulator der Entwicklungsumgebung war allerdings nicht wirklich hilfreich, da dieser sehr viel Leistung vom PC erwartete und die PC's der Gruppenmitglieder dies kaum leisten konnten. Dadurch musste immer ein Smartphone mit einem Kabel am PC angeschlossen sein, damit die App getestet und ausprobiert werden konnte.

Die Erstellung der App mithilfe dieser IDE verlief meist reibungslos, auch aufgrund der sehr guten Integration mit GitLab. Die Gruppenmitglieder konnten sehr gut gleichzeitig an dem Projekt arbeiten, auch wenn es manchmal zu Problemen mit dem "Mergen" kam. Diese Probleme waren aber eher kleinerer Natur und hielten sich in Grenzen.

6.3.4 Layout und Funktionen

Im Folgenden wird auf die einzelnen Screenshots und damit verbunden Activitys der App eingegangen sowie die Funktionen erläutert.

DataHolder

Zum besseren Verständnis des Codes wird an dieser Stelle einmal unser zentraler "DataHolder" erklärt, bevor auf die einzelnen Activitys übergegangen wird. Dieser wird von fast jeder Activity verwendet.

Der DataHolder ist ein Singleton, an dem die Daten gespeichert werden, die mit dem Server synchronisiert werden. Da MQTT verwendet wird und das im Standard Synchronisationsservice von Android nicht vorgesehen ist, wurde mit dem DataHolder eine eigene Lösung implementiert. Per DataHolder.getInstance() kann die Instanz überall in der App genutzt werden. An der Instanz können dann die weiteren getter für das Holen von Daten aufgerufen werden. Das Setzten der Daten erfolgt über die setter Methoden an der Instanz.

Da die Daten nicht von Activity zu Activity weitergeben werden sollten, hat man sich für diese Art der Implementierung entschieden. Da nicht von mehreren Threads gleichzeitig auf das Singleton zugegriffen wird, gibt es keine Probleme auf Grund der Synchronität.

Login-Screen

Um die App nur für Zielrichter mit den nötigen Zugriffsrechten nutzbar zu machen, wird eine Login Activity benötigt (siehe Abbildung 25).

Hier kann sich jeder Benutzer in die App einloggen. Dafür wird ein sechsstelliger numerischer Authentifizierungscode benötigt, welcher vom Kontrollprogramm generiert wird und dort ablesbar ist. Bei Falscheingabe des Codes oder keiner Verbindung zum Kontrollprogramm wird eine entsprechende Fehlermeldung ausgegeben.

Durch die drei Punkte oben rechts kann man auf die Einstellungsseite der App. Dort kann zum Beispiel eingestellt werden, welche IP der Broker hat. Sobald ein Benutzer sich mit einem Code einmal erfolgreich eingeloggt hat, wird dieser gespeichert. Das heißt beim erneuten Starten der App ist der letzte erfolgreich eingegebene Code als Standard vorgegeben.

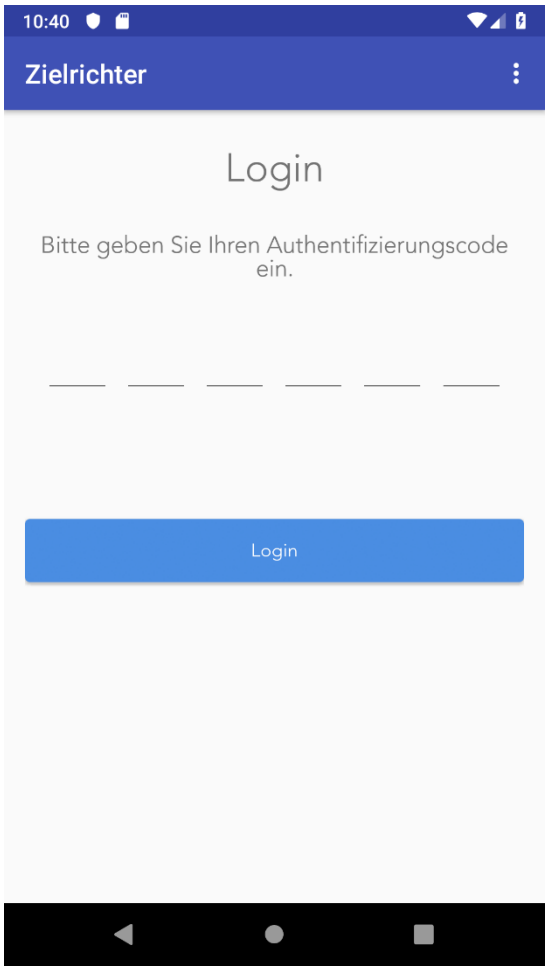


Abbildung 25 Screenshot Login

Funktionalität Login-Screen

Der Login wurde ebenfalls mithilfe der Mqtt Kommunikation realisiert. Sobald der Benutzer einen Code eingegeben hat, wird das Topic **event/referees/configs/[TOKEN]** subscribed. Der Platzhalter [TOKEN] wird dann durch den eingegebenen Code ersetzt. Da die Kommunikation asynchron erfolgt, muss gewartet werden, bis der Server auf die Anfrage geantwortet hat. Sobald die Antwort vom Server ankommt wird das IMqttSubscribeListener messageArrived() Event an der Login Activity ausgelöst. In dieser Nachricht muss die Config des Zielrichers hinterlegt sein, dann wissen wir, dass der Benutzer erfolgreich authentifiziert wurde. Wenn nach 5 Sekunden keine Antwort vom Server erhalten wurde, ist der Login fehlgeschlagen und eine entsprechende Meldung wird ausgegeben. Das Warten von 5 Sekunden wird mit einem Timer realisiert, der nach Auslösen des Loginvorgangs gestartet wird.

Wenn beim Auslösen des Loginvorgangs keine Internetverbindung besteht wird am IMqttSubscribeListener des Event noConnection() ausgelöst. In der Login-Activity wird dann eine entsprechende Meldung angezeigt.

Beim erfolgreichen Login wird die Konfiguration vom Benutzer, die vom Server erhalten wurde, am DataHolder gespeichert. Dadurch sind die Benutzerdaten in der ganzen App verfügbar. Zugleich wird eine entsprechende Bestätigung an den Server gesendet. Dadurch weiß der Kontrollrechner, dass der Benutzer nun online ist und erfolgreich eingeloggt wurde. Ab diesem Moment beginnt die App auch mit dem Senden eines Pings. Diesen Ping sendet die App dann alle 5 Sekunden an den Sever. Dadurch weiß der Server das ein Benutzer noch aktiv und online ist.

Broker Einstellungen-Screen

Um die IP-Adresse auch in der App einstellen zu können, wird diese Activity benötigt (siehe Abbildung 26). Hier ist es dann einfach möglich die IP-Adresse des Kontrollprogramms einzugeben und diese dann zu speichern.

Abbildung 26 Screenshot Broker Einstellungen

Funktionalität Broker Einstellungen-Screen

In dieser Activity kann die IP-Adresse des Brokers vor dem Login eingegeben werden. Nach der Eingabe wird beim Klicken des Buttons ”Speichern” die IP-Adresse des Brockers im DataHolder gesetzt und in einer Datei gespeichert.

Durch das Speichern in einer Datei wird bei erneutem Öffnen dieser Activity die zuletzt eingegebene IP-Adresse aus der Datei ausgelesen und bereits in dem EditText-Feld angezeigt. Das Auslesen und wieder anzeigen geschieht immer beim Öffnen der Activity in der onCreate()-Methode, das Setzen der IP-Adresse immer bei Klick auf den Button ”Speichern” durch setOnClickListener().

Platzierungs-Screen

Diese Activity bildet die Hauptfunktionalität der App ab. Es werden auf der linken Seite die Plätze und auf der rechten Seite die Bahnen angezeigt (siehe Abbildung 27).

Die dunkelblau gefärbten Bahnen entsprechen dabei den belegten Bahnen und die grau gefärbten Bahnen sind nicht belegt.

Zusätzlich dazu wird auch immer der zu bearbeitende Wettkampf und die Laufnummer angezeigt.

Um jetzt die Bahnen auf die Plätze zu positionieren, können die Bahnen einfach per Drag and Drop auf die Plätze gezogen werden. Dabei können auf einem Platz mehrere, gar keine oder nur eine Bahn stehen.

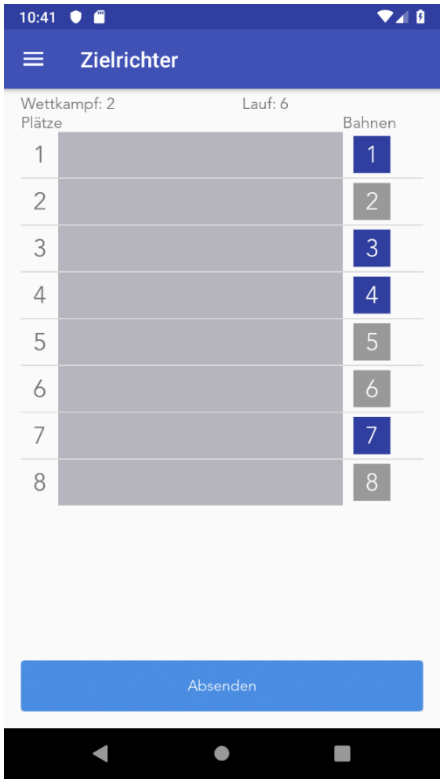


Abbildung 27 Screenshot Platzierungen

Hier links (Abbildung 28) sieht man einmal die Abgabe mehrerer Bahnen auf einer Position, das Freilassen eines Platzes und das Setzen einer nicht belegten Bahn.

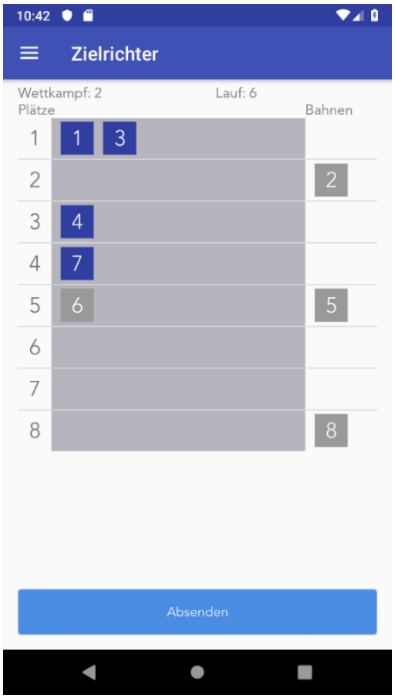


Abbildung 28 Screenshot Platzierungen mit gesetzten Plätzen

Wurde die Abgabe abgesendet, aber noch kein neuer Lauf gestartet, dann erscheint auf der Platzierungs-Seite die Meldung ”Warten auf neuen Lauf” (siehe Abbildung 29).
Sobald der nächste Lauf dann vom Kontrollrechner geschickt wurde, erscheint hier die neue Übersicht.

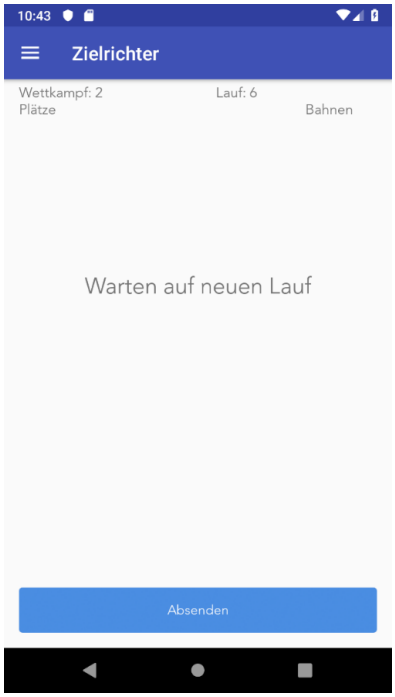


Abbildung 29 Screenshot Platzierungen "Warten auf neuen Lauf"

Funktionalität Platzierungs-Screen

Das Platzierungsfragment ist ein Fragment der MainActivity, welches auch dort erzeugt wird und über das Menü aufrufbar ist.

Bei der Erzeugung des Fragments werden zwei TextViews für die aktuelle Wettkampf- und Laufnummer erstellt, eine ListView für die Plätze mit grauem Bereich zum Ablegen der Bahnen und eine ListView für die Anzeige aller Bahnen mit farbiger Markierung, welche Bahnen belegt sind und welche nicht. Die ListView für die Plätze werden zu Beginn mit den Plätzen 1 bis 8 initialisiert, da es in Schwimmhallen für Wettkämpfe in der Regel acht Bahnen geben kann. Daher wird auch die ListView für die Bahnen mit den Bahnen 1 bis 8 initialisiert.

Ein Button zum Absenden der Ergebnisse wird ebenfalls erstellt. Dieser speichert beim Anklicken die abgegebene Reihenfolge in der Klasse RefereeResult mit der zugehörigen Zielrichter ID, um die Ergebnisse eindeutig den Zielrichtern zuordnen zu können. Da die Ergebnisse aber noch nicht an den Kontrollrechner gepublished werden sollen, sondern erst an die nachfolgende Activity (Abgabebestätigung) übergeben werden sollen, muss es als Intent im JSON Format an diese übergeben werden. Eine andere Art des Datenaustauschs zwischen zwei Activity ist nicht möglich und muss daher über den Intent im JSON Format erfolgen.

Um das Drag und Drop zu realisieren, wurden auf dem Platzierungsfragment innerhalb eines linearen Layouts zwei ListViews platziert. Ein ListView repräsentiert die belegten Bahnen, während die andere ListView die Platzierung anzeigt. Während der Initialisierung wird für die ListView für die Bahnen, (listViewBahnen), ein DragListener gesetzt. Dieser DragListener implementiert das Interface View.OnDragListener in welchem die Methode onDrag() überschreiben wird. Die listViewBahnen ist somit nun „dragable“ und es muss implementiert werden, welche Aktionen danach durchgeführt werden sollen. Dies wird über bestimmte DragEvents realisiert, die der onDrag()-Methode übergeben werden. Somit ist der onDrag()-Methode bekannt welche View vom Nutzer ausgewählt wurde und es kann festgelegt werden, bei welchem Event, wie reagiert werden soll. Für das klassische Drag and Drop ist deshalb das ACTION_DROP-Event zu verwenden. Innerhalb dieses Case-Blocks wird ermittelt welche TextView ausgewählt wurde und auf welches Layoutobjekt die View gedroppt werden soll. Die View wird dann dementsprechend aus der einen ListView entfernt und in die andere ListView hinzugefügt.

Damit immer die aktuelle Wettkampf- und Laufnummer angezeigt wird, gibt es noch die Methode updateCurrentRace(). Dort wird immer die aktuelle Wettkampf- und Laufnummer gesetzt. Zusätzlich wird noch überprüft, ob für den aktuellen Wettkampf bereits etwas abgegeben wurde. Wenn dies der Fall ist, dann werden die beiden ListViews zur Anzeige der Plätze und Bahnen unsichtbar gemacht und der Text “Warten auf nächsten Lauf” sichtbar gemacht (Siehe Abbildung 29). Der Button zum Absenden wird dann auch gesperrt, damit nicht fälschlicherweise etwas abgegeben werden kann. Wenn für den aktuellen Lauf noch nichts abgegeben wurde, wird die nächste Übersicht mit den neuen farbig markierten belegten Bahnen angezeigt und der Button zum Absenden freigegeben.

Ob eine Bahn belegt ist, also blau dargestellt wird, oder ob sie nicht belegt ist und damit grau angezeigt wird, überprüft die BahnenListAdapter-Klasse in der Methode getView(). Dort werden die belegten Bahnen aus dem aktuellen Wettkampflauf aus dem DataHolder geholt und dadurch alle nicht belegten Bahnen bestimmt und diese dann grau eingefärbt.

Abgabebestätigungs-Screen

In diesem Fenster wird nach der Abgabe einmal die eingegebene Abgabe angezeigt (siehe Abbildung 30).

Hier können die Zielrichter ihre Abgabe dann noch einmal überprüfen und falls sie noch etwas korrigieren möchten, kann über den Zurück-Button in das Platzierungsfenster zurückgewechselt werden. Die Abgabe wird dann dort auch wieder angezeigt, sodass sie es einfach noch einmal nacharbeiten können.

Wenn die Zielrichter dann ihre Eingabe überprüft haben, können sie über den Button Absenden ihre Eingabe endgültig abgeben.

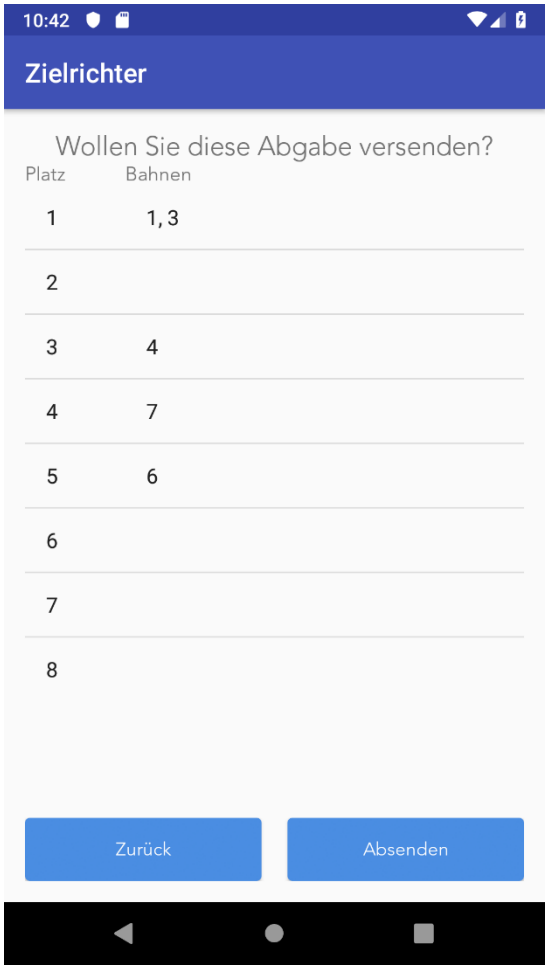


Abbildung 30 Screenshot Abgabebestätigung

Funktionalität Abgabebestätigungs-Screen

Diese Aktivität verwendet die Klassen RefereeResult und CurrentRace. Bei der Erzeugung werden mehrere ListViews und Buttons erstellt. Eine ListView beschreibt die Bahnen und eine weitere ListView beschreibt die Plätze. Über das Intent wird sowohl das abgegebene Ergebnis als auch der aktuelle Lauf wiederhergestellt. Zwischen Aktivitäten kann nur über ein Intent Daten ausgetauscht werden. Die vom Zielrichter zuvor festgelegte Platzierung wird ausgewertet und über die ListViews werden gesetzt, um das Abgabeergebnis nochmal anzuzeigen.

Der Button zum Absenden löst die Methode send() aus. In dieser Methode wird die Race-ID und die Contest-ID für die Klasse RefereeResult gesetzt, damit die Abgabe dem aktuellen Lauf zugewiesen wird. Diese Klasse wird dann auf das Topic „event/referees/raceResult“ gepublished.

Wenn das Ergebnis erfolgreich gepublished wurde, wird das deliveryComplete() ausgelöst. Dort wird gespeichert, welches der zuletzt gesendete Lauf war und dieser wird per intent.putExtra() ebenfalls gesichert, damit andere Aktivitäten wie z.B. die Abgabenübersichtsaktivität dies auswerten können.

Weiter-Screen

Nach der Abgabe der Ergebnisse wurde in einer älteren Version der App auf eine Weiter-Seite weitergeleitet (siehe Abbildung 31).

Diese hatte jedoch nur zur Folge, dass die Zielrichter einmal mehr klicken mussten und somit etwas aufgehalten hat.

Nach dem Testeinsatz wurde diese Seite dann entfernt. Einige Zielrichter hatten uns während des Testeinsatzes auf diese Änderung hingewiesen.

Abbildung 31 Screenshot Weiter

Abgabenübersicht-Screen

In dieser Activity wird angezeigt, welche der Zielrichter bereits abgegeben haben und welche noch nicht. Sie wird jetzt direkt nach der Abgabenbestätigungsseite angezeigt (siehe Abbildung 32).

Zusätzlich hat der Zielrichter die Möglichkeit den Lauf zu beenden, falls einer der Zielrichter nicht abgeben kann. In diesem Screen hier sind noch Beispieldaten eingefügt, aber im echten Betrieb stehen dort die IDs der einzelnen Zielrichter. Die jeweilige ID ist im Menü Zielrichterobmann und Menü Zielrichtereinzusehen.

Klickt der Master den Lauf beenden Button erfolgt eine Sicherheitsabfrage. Auch diese Abfrage wurde erst nach dem Testeinsatz eingefügt, da es beim Adventsschwimmen vorkam, dass dieser Button versehentlich geklickt wurde.

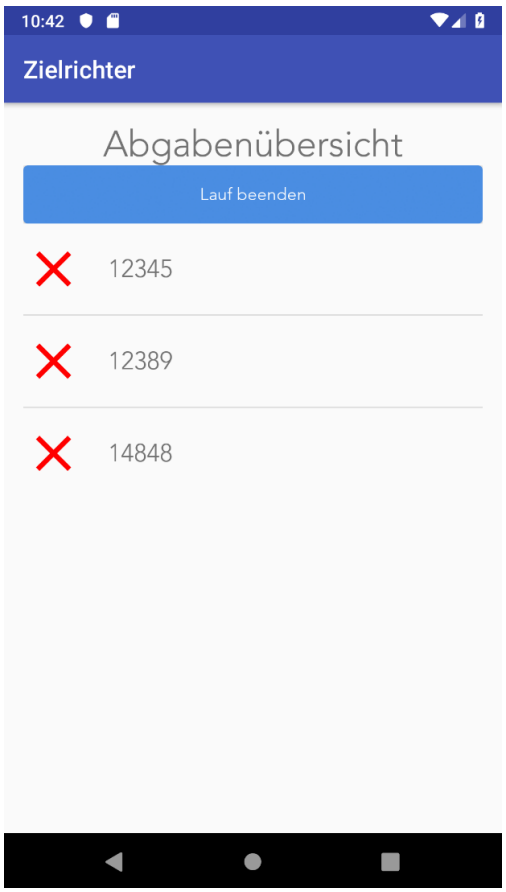


Abbildung 32 Screenshot Abgabenübersicht

Aus der Sicherheitsabfrage kann entnommen werden, ob der Lauf tatsächlich beendet werden soll (siehe Abbildung 33).

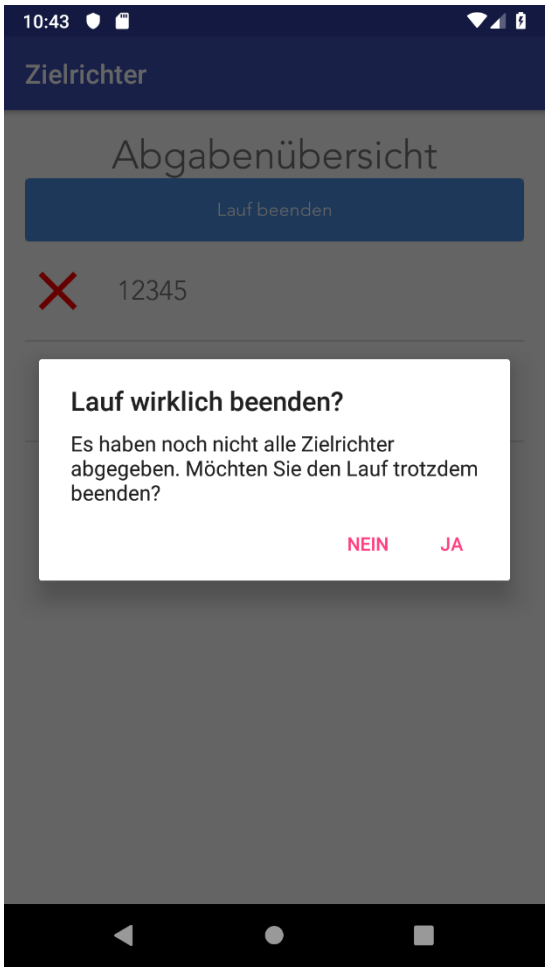


Abbildung 33 Screenshot Sicherheitsabfrage Abgabenübersicht

Funktionalität Abgabenübersicht-Screen

In der onCreate()-Methode dieser Aktivität wird mithilfe des Intents die gespeicherte Abgabe ausgewertet. Handelt es sich dabei um den letzten Lauf wird die Aktivität direkt beendet, damit in der Anwendung der nächste Lauf bearbeitet werden kann. Die Intent-Klasse ist eine Klasse von Android, um Informationen zwischen einzelnen Aktivitäten auszutauschen. Da die Abgabe bereits feststeht, auf einer anderen Aktivität ermittelt und bereits versendet wurde, muss über das Intent dies an die Abgabenübersicht weitergeben werden.

Es wird ein ListView erzeugt, in welchem die Informationen angezeigt werden, welcher Zielrichter bereits abgegeben hat. Im onCreate() werden dabei noch Dummy-Einträge angelegt. Diese werden später überschrieben, sobald die korrekten Informationen an den Zielrichter gesendet wurden. Zudem wird noch der Button erzeugt, um den Lauf zu beenden. Dieser löst die entsprechende Sicherheitsabfrage aus und beendet ggf. den Lauf.

Es wird sich auf das Topic „event/referees/resultOverview“ subscribed. Sobald der MQTT-Server für diese Topic eine Änderung published, wird die messageArrived()-Methode aufgerufen. In der Methode messageArrived() wird der zugesendete String ausgewertet. In diesem sind die Informationen enthalten, welcher Zielrichter bereits abgegeben hat. Diese Information wird in der dazugehörigen Klasse ResultOverview gesichert und daraufhin werden die Ergebnisse durchgegangen und in einer ArrayList gespeichert. Diese ArrayList enthält die IDs der Zielrichter und ob diese abgegeben haben oder nicht. Über einen ZielrichterListAdapter, welcher die ArrayList als Parameter bekommt, wird die

ListView neu gesetzt und aktualisiert. Sobald vom MQTT-Server gepublished wird, dass ein weiterer Zielrichter abgegeben hat, wird diese Information erneut ausgewertet und das ListView wird aktualisiert.

Menü Zielrichterobmann und Menü Zielrichter

Je nach Rolle des Zielrichters ist ein anderes Menü zu sehen. Der Zielrichterobmann (Master) hat nämlich die Möglichkeit den Wettkampf zu wechseln (siehe Abbildung 34. Sonst unterscheidet sich nichts (siehe Abbildung 35). Im Menü kann außerdem die Zielrichter ID eingesehen werden. Vom Menü aus kommt man auf die Platzierungsseite sowie den Zielrichterstatus und der Obmann zusätzlich noch auf die Wettkampfauswahl. Auch können sich die Zielrichter hier abmelden.

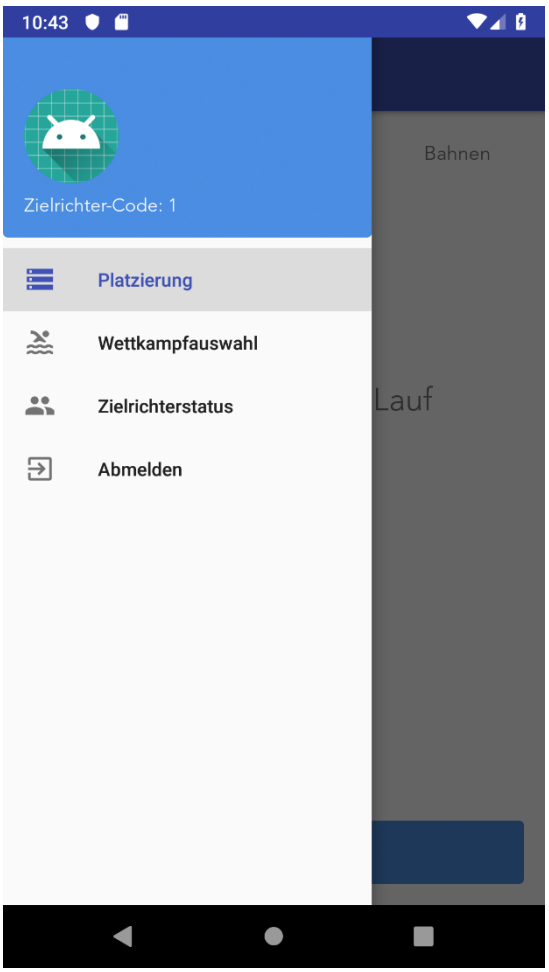


Abbildung 34 Screenshot Menü Master-App

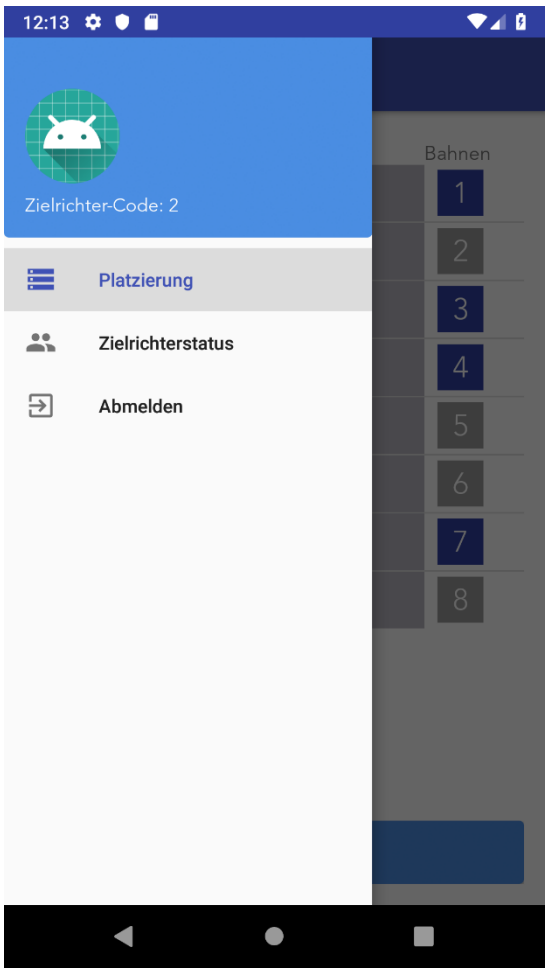


Abbildung 35 Screenshot Menü normaler Nutzer

Logout

Die Logout Funktion wird benötigt, um sich aus der App auszuloggen. Nach dem Logout wird auf die Login Seite gesprungen, wo die IP-Adresse des Kontrollrechners (Brokers) geändert werden und sich mit einem generiertem Authentifizierungscode wieder eingeloggt werden kann.

Funktionalität Logout

Die Funktionalität des Logouts ist in der Klasse MqqtCommunicator implementiert und wird in der MainActivity aufgerufen, wenn auf abmelden getippt wurde.

Bei einem Logout wird auf das Topic “event/referees/loggedOut/[TOKEN]” ein “true” gepublished um dem Kontrollrechner zu signalisieren, dass sich ein bestimmter Zielrichter ausgeloggt hat. Zusätzlich

müssen alle SubscribeListener abgemeldet werden, damit nicht mehr auf die Topics subscribed wird. Auch die Verbindung zum Kontrollrechner (Broker) muss getrennt werden, um eine erneute Verbindung wieder möglich zu machen.

Danach werden alle Listener und bestimmte Flags neu erstellt und die aktuelle Activity, in der sich die App befindet, geschlossen. Danach wurde sich erfolgreich aus der App ausgeloggt.

Zielrichterstatus-Screen

In diesem Screen wird angezeigt, welche Zielrichter online sind (siehe Abbildung 36). Dabei werden analog zur Abgabenübersicht wieder die IDs der Zielrichter angezeigt. In diesem Fall sind die Zielrichter mit den IDs 1 und 3 online und der Zielrichter mit der ID 2 ist offline.

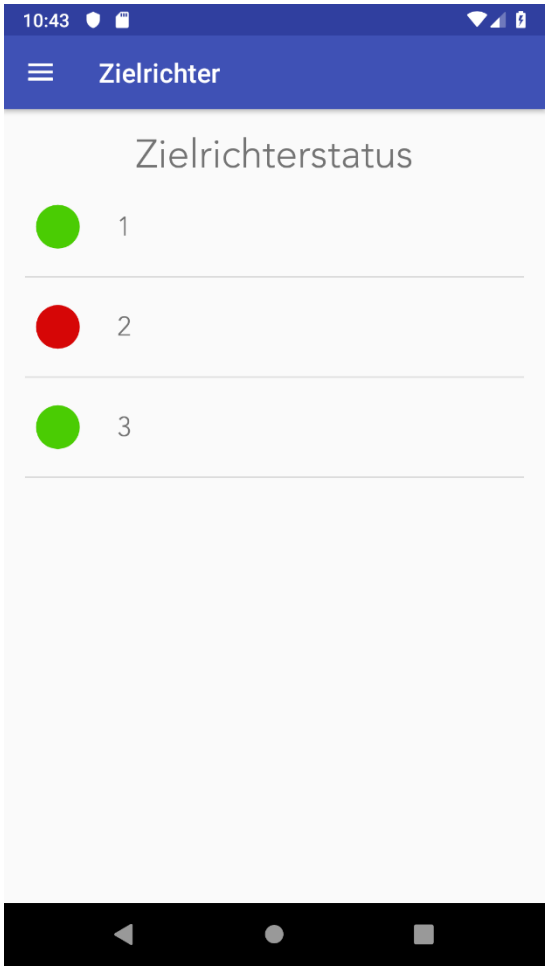


Abbildung 36 Screenshot Zielrichterstatus

Funktionalität Zielrichterstatus-Screen

Der Zielrichterstatus wurde über ein Fragment, welches in der MainActivity erzeugt und aufgerufen wird, realisiert. Ein Fragment ist ein modularer Bereich einer Aktivität.

Analog zur Aktivität für die Abgabenübersicht wird bei Erzeugen des Fragments eine ListView mit Dummy-Daten erstellt und es wird sich auf das Topic „event/referees/status/statusOverview“ subscribed. In der MessageArrived()-Methode wird die Message in der Klasse StatusOverview gesichert. Für jeden Eintrag wird die Information, ob ein Zielrichter online oder offline ist zu einer ArrayList hinzugefügt und dies wird an einen ZielrichterListAdapter übergeben. Dieser ZielrichterListAdapter wird an dem ListView neu gesetzt und die ListView kann aktualisiert werden. Der MQTT-Server sendet somit allen Zielrichtern, welche Zielrichter ebenfalls online bzw. offline sind.

Bei Programmstart wird ein Thread gestartet, der im Hintergrund arbeitet. Dieser Thread published alle 5 Sekunden auf das Topic „event/referees/status“, dass er noch aktiv ist. Beim Beenden des Programmes oder beim Ausloggen wird dieser Thread gestoppt. Der Kontrollrechner wertet den

Zielrichterstatus aus und gibt die Information, welche anderen Zielrichter ebenfalls online sind wieder zurück.

Wettkampfauswahl-Screen

In diesem Screen kann der Zielrichterobmann den aktuellen Wettkampf und Lauf ändern (siehe Abbildung 37). Dazu wird zunächst die Wettkampfnummer ausgewählt und danach die Laufnummer, so werden auch immer nur die passenden Laufnummern zu dem jeweiligen Wettkampf angezeigt.

Zusätzlich dazu wird einmal der aktuelle Wettkampf mit Bezeichnung und aktueller Laufnummer angezeigt und dann auch die nächsten Wettkämpfe und Läufe.

Hat der Obmann einen Wettkampf und Lauf gewählt, kann dieser mit Klick auf den Start-Button gestartet werden. Jede Zielrichter-App wird jetzt wieder auf den Platzierungs-Screen weitergeleitet, egal auf welcher Seite sich die Zielrichter befinden.

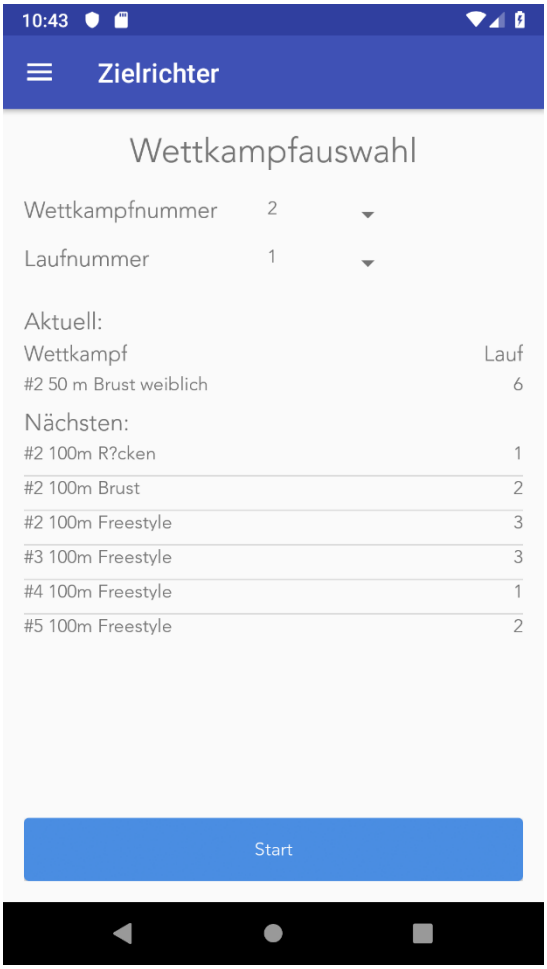


Abbildung 37 Screenshot Wettkampfauswahl

Funktionalität Wettkampfauswahl-Screen

Die Wettkampfauswahl ist ebenfalls ein Fragment der MainActivity, welches auch dort erzeugt wird. Es ist über das Menü aufrufbar.

Beim Erzeugen dieses Fragments werden zunächst zwei Spinner zum Auswählen der Wettkampf ID und Lauf ID erstellt. Zusätzlich wird zur Anzeige der nächsten Wettkämpfe und Läufe eine ListView erstellt, die die nächsten Wettkampf IDs und Lauf IDs enthält. Beim Aufrufen der Activity werden die nächsten Wettkampfnummern, Wettkampfnamen und Laufnummern aktualisiert und in die ListView geschrieben. Auch der Button zum Starten des ausgewählten Laufs wird beim Erzeugen des Fragments erstellt. Dieser Button published beim Drücken die eingestellte Wettkampf- und Laufnummer auf das Topic “event/referees/raceData”. Zusätzlich wird sich auch noch auf das Topic “event/referees/contestData” subscribed.

Wenn jetzt eine Message vom Kontrollrechner auf das Topic “event/referees/contestData” gepublished wird, dann wird in der MessageArrived()-Methode die Message in der Klasse contestData gesichert und die ListView mithilfe eines Adapters mit den nächsten Wettkämpfen und Läufen gefüllt. Der Spinner mit den Wettkampf IDs wird hier auch mit den gesendeten Wettkampf IDs gefüllt. Dabei wird ein TreeSet verwendet, um die IDs in aufsteigender Reihenfolge anzuzeigen. Dieser Spinner erhält auch einen onItemSelectListener, um bei der Auswahl einer Wettkampf ID die passenden

auswählbaren Lauf IDs in dem unteren Spinner anzuzeigen. So wird sichergestellt, dass keine Wettkampf-Lauf Kombinationen auftreten, die gar nicht existieren.

Wird jetzt eine Wettkampf ID ausgewählt, dann wird die Methode onItemSelected() aufgerufen. Dort wird zunächst geprüft, ob eine Wettkampfnummer oder eine Laufnummer ausgewählt wurde. Wenn eine Wettkampfnummer ausgewählt wurde, dann werden zu der ausgewählten Wettkampfnummer alle passenden Laufnummern in den Spinner für diese eingetragen und der Spinner bekommt noch einen onItemSelectListener. Auch wird die Wettkampf ID dann auf die ausgewählte ID gesetzt. Falls die Lauf ID ausgewählt wird, dann wird diese auf die ausgewählte ID aktualisiert.

6.3.5 MQTT Kommunikation

Neben der Definition der MQTT-Topics und der eigentlichen Programmlogik ist ein weiterer wichtiger Bestandteil die Kommunikation über MQTT.

Um MQTT für Android nutzen zu können, wurde das open-source Projekt Paho verwendet. Dies stellt bereits Klassen und Interfaces zur Verfügung, die für die eigene MQTT-Kommunikation genutzt werden können.

Für die Kommunikation wurde eine abstrakte Klasse AbstractMqttCommunicator erstellt. An dieser sind die Methoden, um einen Inhalt zu publishen oder sich auf ein Topic zu subscriben, vorgegeben. Außerdem enthält die Klasse mehrere HashMaps, mit denen die Verarbeitung realisiert wurde. Diese HashMaps enthalten jeweils einen String (das Topic) und den jeweiligen Publish-/Subscribelistener. Von der abstrakten Klasse erben die Klassen MqttCommunicator und MqttCommunicatorMockUp.

Die Klasse MqttCommunicatorMockUp wird dabei ausschließlich zu Testzwecken verwendet. Die eigentliche Implementierung erfolgt in der Klasse MqttCommunicator. Dort wird im Konstruktor eine Verbindung zum verwendeten MQTT-Server hergestellt. Außerdem sind dort die Methoden subscribeTopic() und publishTopic() ausprogrammiert, welche dafür sorgen, dass die eigentliche Kommunikation stattfindet. Mithilfe der von Paho bereitgestellten Klasse MqttAndroidClient kann dann das eigentliche Protokoll verwendet werden. Die eigentliche Kommunikation mithilfe dieser Klassen realisiert.

Eine weitere Klasse MqttConnection sorgt dafür, dass es immer nur eine aktuelle Instanz im Programm gibt, die für die Kommunikation zuständig ist. Das Singleton MqttConnection enthält als private Variable den MqttCommunicator. Dieser kann geholt und gesetzt werden und innerhalb des Programms kann über den Aufruf MqttCommunicator.getInstance(), die aktuelle Instanz verwendet werden. Somit ist sichergestellt, dass es innerhalb des Programmes nur eine Instanz gibt, welche die eigentliche Kommunikation regelt.

Um die Nutzung der Topics und Inhalte so einfach und so erweiterbar wie möglich zu gestalten, wurden zwei Hilfsklassen MqttSubcriber und MqttPublisher erstellt. Diese beiden Klassen enthalten statische Methoden für jedes Topic, um auf dieses zu subscriben bzw. zu publishen. Als Parameter ist dort lediglich der eigentliche Listener notwendig und gelegentlich der Token, wenn es sich um ein zielrichterspezifisches Topic handelt. Der Listener ist in diesen Zusammenhang die Aktivität, welche ein Event erhalten möchte, wenn sie etwas erfolgreich published hat oder sich ein Wert auf einem der subscribten Topics ändert.

Um diesen Austausch zu gewährleisten, wurden zwei Interfaces erstellt (IMqttSubscribeListener und IMqttPublishListener). Diese geben die Methoden vor, welche ausgeführt werden, wenn eine Kommunikation mit dem Server abläuft. Zum Beispiel enthält das Interface IMqttSubscriberListener die Methode messageArrived(). Diese Methode wird ausgelöst, sobald vom Server ein Inhalt auf ein

bestimmtes Topic gepublished wird. Eine Aktivität, die sich auf ein bestimmtes Topic subscriben möchte, implementiert das Interface IMqttSubscribeListener und überschreibt die Methode messageArrived(). Anhand der Parameter topic und message, kann an der Aktivität nun ermittelt werden, welches Topic einen neuen Inhalt erhalten hat und um welchen Inhalt es sich handelt. Somit ist auch möglich, dass eine Aktivität sich auf beliebig viele Topics subscribed. Zudem stehen noch weitere Methoden wie z.B. onSuccess() oder onFailure() zur Verfügung, um zu überprüfen, ob die Kommunikation erfolgreich ist. Das ermitteln, welche Listener sich für welche Topics subscribed haben oder an welches Topic etwas publishen wollen, wird dabei vom MqttCommunicator mithilfe der HashMaps verwaltet. Ebenso wird der jeweilige Aufruf an diesen Stellen geregelt, sodass eine Aktivität lediglich die Interfaces implementieren und die Methoden überschreiben muss, um vollständig mit MQTT kommunizieren zu können. Wenn ein weiteres Topic hinzugefügt werden muss, können an den Hilfsklassen MqttSubcriber und MqttPublisher weitere statische Methoden hinzugefügt werden. Diese können dann an den jeweiligen Aktivitäten verwendet werden, um sich auf das neue Topic anzumelden oder dort etwas zu publishen. Damit die Informationen nicht doppelt ankommen, wird beim Beenden jeder Activity ein unsubscribe auf alle abonierten Topics dieser Activitis gemacht.

Die Informationen der Listener wird solange gespeichert, bis sich der Nutzer selbstständig ausloggt oder er vom Kontrollrechner ausgeloggt wird. Bei einem Logout werden sämliche HashMaps, Flag etc. zurückgesetzt, damit sichergestellt wird, dass bei einem Wechsel des Nutzers keine fehlerhaften Zuweisungen mehr gespeichert sind.

6.3.5.1 MQTT-Kommunikation mit dem Kontrollrechner

Der Kontrollrechner fungiert innerhalb des Netzwerkes als MQTT-Server. Somit melden sich alle beteiligen Clients bei diesem an.

Um die einzelnen Zielrichter zu identifizieren, wurde ein sogenannter Token verwendet. Dieser wird vom Kontrollrechner erzeugt und somit ist dem Server immer bekannt, welcher Zielrichter z.B. eine Abgabe oder eine Statusmeldung gesendet hat. Bevor die Zielrichter-App mit dem Server kommunizieren kann, muss bei Programmstart dieser Token ausgetauscht werden. Der Token beschreibt gleichzeitig den Login-Code und es wird geprüft, ob es ein entsprechendes Topic für diesen Token gibt. Dieses Topic und alle weiteren Topics mit relevanten Informationen für die Zielrichter-App müssen vor der Nutzung der Applikation vom Kontrollrechner entsprechend bereitgestellt werden (z.B. raceData oder contestData).

Nachdem der Tokenabgleich und somit die Anmeldung erfolgreich war, subscribed sich die Applikation auf die notwendigen Topics und erhält die Informationen, welche Läufe zum Wettbewerb gehören oder welcher Lauf der aktuelle ist. Sobald der Kontrollrechner z.B. den aktuellen Lauf ändert, erhält die Anwendung diese Information, kann dementsprechend reagieren und den Lauf intern ebenfalls neu setzen.

Sobald ein Lauf von der Zielrichter-App abgegeben wird, wird diese Information auf das dazugehörige Topic raceResults gepublished. In solchen Fällen ist der Kommunikationsweg umgedreht, da die Zielrichter-Applikation einen Inhalt published. Der Kontrollrechner erhält die Information, dass ein neuer Inhalt verfügbar ist. Für jedes Topics ist vorab definiert worden, welche Anwendungen sich subscriben und welche Anwendung etwas publishen müssen.

6.4 Datenbank

Hinweis: Die Datenbank war ursprünglich geplant und hat sich im Verlauf des Projektes als impraktikabel erwiesen. Weiteres zur Entscheidung finden Sie im Unterkapitel 6.4.2. Verwurf der Datenbank.

6.4.1. Spring / JPA / Hibernate

Die Verwendung von Spring Boot ([9]) bietet hierbei in Verbindung von JPA große Vorteile bei der Arbeit mit der Datenbank. Spring Boot kümmert sich um die Konfiguration und Anbindung der Datenbank und es muss nur ein minimaler Konfigurationsaufwand getätigt werden. Es ist also möglich, dass zwischen Datenbankmanagementsysteme mit nur 1-2 Zeilen gewechselt werden kann. Die Java Persistance API (kurzform JPA) ist eine Schnittstelle für die Datenbank und der Business Logic der Anwendung. Anstelle mühsam eine eigene API für die Datenbankkommunikation zu schreiben ist JPA eine große Hilfe, da bei der Erstellung dieser API ein großer repetitiver Aufwand darstellt – es befindet sich an vielen Stellen der identische Code bei einer eigenen API. Ebenfalls ein großer Nachteil der Erstellung einer eigenen API ist die hohe Fehleranfälligkeit. Es können sich viele gefährliche Bugs einschleichen, was mehr Managementaufwand impliziert. Der größte Nachteil ist. dass eine eigene API bei weitem zeitaufwändiger als die Verwendung von JPA ist.

Das besondere an JPA / Hibernate ist die Generierung der API. Ein bereits vorhandenes Datenbankschema wird genommen und für jede Relation eine eigene Entity-Class angelegt mit all den Attributen, Eigenschaften und Beziehungen zwischen den Relationen. Die Erstellung eines Datenbankschema erfolgte bei unserem Projekt innerhalb von MySQL Workbench ([10]) als .erm-Datei, welche jederzeit von MySQL Workbench als Datenbankschema transformiert wird.

In Eclipse direkt ist es möglich die Entity-Klassen von einem Datenbankabbild zu generieren. Mittels Repositories ist es möglich, eine Menge an Entities zu erhalten mit welche innerhalb von Java gearbeitet wird. Während Entitites eine abstrakte Darstellung einer Relation sind, erfüllen Repositories den Zweck des Datenspeichers. Es können neue Entities innerhalb eines Repositories angelegt werden, vorhandene Rows einer Relation via find…(…) geladen werden. Ebenfalls können vom Entwickler selbstdefinierte SQL-Befehle verwendet werden. Weiteres zu Repositories und Entities befindet sich auf den respektiven Hilfeseiten.

6.4.1.1. Entity

```
@Entity
@Table(name="swimclub")
@NamedQuery(name="Swimclub.findAll", query="SELECT s FROM Swimclub s")
public class Swimclub implements Serializable {
	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	@Column(unique=true, nullable=false)
	private int swimclubID;
	@Column(length=45)
	private String name;
	//bi-directional many-to-one association to Swimmer
	@OneToMany(mappedBy="swimclub")
	private List<Swimmer> swimmers;
	public Swimclub() {
	}
	// Getter und Setter…
...
}
```

Listing 1: JPA Entity-Klasse für Schwimmclubs

Dies ist eine Entity-Klasse, welches das Abbild der Relation „Swimclub“ darstellt. Es werden hierbei Annotations verwendet, die JPA als solche interpretiert. @Entity, @Table und @NamedQuery weißen darauf hin, dass es sich hierbei um ein Model handelt von der Relation swimclub auf wie mit dem

findAll-Query (relevant für Repositories) umgegangen wird. Da ein Entity vom Inteface Serializable implementiert, werden alle Attribute (ohne transient-Keyword) als Column in der Relation abgebildet. Mittels den angegeben Annotations wird die Eigenschaft der Column bestimmt – variable Längen, Primary Key, Auto Increment, Beziehungen zu anderen Relationen um ein paar zu nennen.

6.4.1.2. Repository

```
public interface PersonRepository extends CrudRepository<Person, Integer> {
    Optional<Person> findByFirstnameAndAftername(String firstname, String
afternamе);

}
```

Listing 2: JPA Repository-Interface für Personen-Entities.

Repositories (grundsätzlich Interfaces) erben von spezifischen JPA-Interfaces - in dem Fall CrudRepository. Das Interface besitzt 2 generische Parameter. Der 1. bestimmt für welches Entity das Repository gekoppelt ist. Der Primärschlüsseldatentyp wird im 2. generischen Parameter angegeben. Methoden innerhalb des Interfaces führen im Hintergrund von Java via Hibernate SQL-Befehle aus. In Repositories müssen entweder strenge Methodennamen vergeben werden oder mittels @Query-Annotation eigene SQL-Befehle angegeben werden.

Bei dem Beispiel weiß JPA, dass eine Person zurückgegeben wird, wenn der Vor- und Nachname gemäß den Parametern in der Datenbank gefunden wird. Andernfalls wird zwar ein Objekt vom Typ Optional zurückgegeben, aber in dieser befindet sich keine Person. Repositories müssen in einer Spring-Anwendung als Klassenattribut via @Autowired definiert werden. Spring holt sich die Werte zur Laufzeit von der Datenbank in Java rein. Mit dem Objekt des Repository können nun die im Interface deklarierten Operationen ausgeführt werden, sowie die vom Basisinterface – bei CrudRepository wäre es z.B. save für das Hinzufügen & Updaten von Entities, delete zum Entfernen von Entities und findAll um die Liste aller Rows der Relation zu erhalten.

6.4.2. Verwurf der Datenbank

In der 1. Version der Topologie war vorgesehen, für das komplette Projekt eine eigene Datenbank zu haben und diese auf dem Kontrollrechner zu platzieren. Dabei hätte man (unnötigerweise) ein verteiltes System. Eine Komponente der Topologie wäre stark ausgelastet, da diese alle Anfragen der anderen Komponenten zu bearbeiten hat. Dies hat sich jedoch schnell als ziemlich impraktikabel erwiesen und es wurde beschlossen, den Kontrollrechner und den Auswerter-PC jeweils eine eigene Datenbank aufzustellen. Dies war der finale Stand, bis zum Beschluss zur Entfernung der Datenbank.

Zum Ende des Projektablaufes hat sich jedoch rausgestellt, dass die Verwendung der Datenbank sich als nicht sonderlich praktikabel erwiesen hat. Eine Datenbank soll primär Daten effizient aufnehmen & bereitstellen. Die Schnittstelle zwischen Datenbank und dem Kontrollrechner und Auswerter-PC war ein ausschlaggebendes Problem gewesen. Schließlich nutzt JPA eigene Klassen, um mittels Hibernate mit der H2-Datenbank zu kommunizieren. Die Entwickler der beiden Teilkomponenten müssten also ihre kompletten Models, mit dem der Datenbank anpassen. Da es bei diesem Fortschritt einige Probleme gab, insbesondere bei den Relationen (n:m; 1:n; usw…) zu anderen Models, war das Debuggen eine Qual. Dann müssen Datenbankoperationen mittels Repositories realisiert werden. Diese Operationen müssen an jede Stelle geschehen, in welchen sich die Entities geändert haben. In der letzten Version mit Datenbank funktionierte die Anbindung weitestgehend, jedoch nicht das Laden von der Datenbank, welches ebenfalls sehr viel Aufwand darstellen würde. Schließlich arbeiten in diesem Prozedere 2 Teams miteinander, welcher jeweils von dem gegenseitigen Code vorher nicht viel gesehen haben.

Anstelle über die Datenbank Backups durchzuführen, wird von Java direkt die Klassenobjekte als Blobs gespeichert & geladen, welches ziemlich leicht und fehlerfrei erfolgt. Während bei der Datenbank etwaige Meetings stattfand, um diese im Kontrollrechner und dann noch im Auswerter-PC zu integrieren, war die Backup-Funktion mittels Blobs schnell programmiert und zusätzlich ist diese Methode besser zu debuggen und viel weniger fehleranfällig, als im Hintergrund eine Datenbank laufen zu lassen, welche lediglich eine Backup-Funktion aufweist. Und es muss nicht der komplette, funktionierende Code eines Systems komplett für die Backup-Funktionalität überarbeitet werden.

6.5 Lenex Austausch

6.5.1 Import

Wie in der Modellierung beschrieben, wird aus einer Lenex-Datei (Vorlagedatei) die POJO´s generiert ([11]). Es können dabei nun alle Dateien, die dieser Vorlage entsprechen innerhalb von Java interpretiert werden.

```
JAXBContext jc = JAXBContext.newInstance(LENEX_ME.class);
Unmarshaller u = jc.createUnmarshaller();
LENEX_ME lenex = (LENEX_ME)u.unmarshal(file);
```

Listing 3: JAXB kommt im Einsatz und liest eine Lenex-Datei ein und interpretierte diese in Java.

In der Abbildung ist dargestellt, dass das Lenex-Objekt nun jegliche Werte aus den Meldedaten beinhaltet und nun innerhalb des Systems verwendet werden kann. Da man aus den Meldedaten die kompletten Läufe samt Wettkampfinformationen, Schwimmern usw. erhalten möchte, wird eine Liste angelegt, welche durch den Parser befüllt wird. Anschließend kann die Liste für jegliche Operationen von dem Kontrollrechner verwendet werden.

6.5.2. Export

Relevant für den Export sind ist die Anzahl der Zwischenzeiten *(#Zwischenzeiten*) und ab welcher Distanz (über den Parameter *Parameter_ZZ* angegeben) diese aufgenommen werden. Dieser Parameter wird vom Auswerter-PC eingelesen. Sofern keine Angaben erfolgten, beträgt die Default-Distanz 100m. In unserem Algorithmus spielt die Distanz des Wettkampfes eine (*Distance*) eine Rolle. Zusätzlich können Schwimmer in Staffeln schwimmen, somit spielt die Anzahl der Staffelteilnehmer (*#Staffel*n) auch eine Rolle. Die Anzahl der Zwischenzeiten lässt sich durch einen einfachen Algorithmus ermitteln, welcher wie folgt mathematisch lautet:

$$\#Zwischenzeiten = \frac{\#Staffeln \cdot Distance < Parameter\_ZZ ? \begin{cases} ja: Parameter\_ZZ \\ \text{nein: } Distance \end{cases}}{Parameter\_ZZ} - 1$$

Ein Sonderfall ergibt sich, wenn der `Schwimmstil = "RELAY"` & `#Staffeln = 1` & `Distance = 200` & `Parameter_ZZ > 50` ist. Für die Einzelstrecke über 200m Lagen werden keine Zwischenzeiten genommen, wenn die Zwischenzeitdistanz (Parameter_ZZ) mehr als 50m beträgt. Teile des Algorithmus sind bereits bei der Importierung integriert, da *Distance* und *#Staffeln* von der Lenex-Datei eingelesen werden und diese – gemäß der Topologie – lediglich beim Kontrollrechner vorliegt.

Im Auswerter-PC kann mittels des Buttons „Lauf speichern“ die vorbereitete ALGE-Lenex-Datei generiert werden. Diese hat den Aufbau, welcher im Unterkapitel „Lenex (Modellierung): Auswerter zu Lenex/EasyWk“ zu finden ist. Dabei wird die Methode `parseToLenex(int heatid, List<Swimmer2Race> race)` aufgerufen. Es wird hierbei eine Datei <HeatID>.txt angelegt,

welche anschließend vom FileChangeListener (siehe nächstes Unterkapitel) zum Export in das Programm EasyWK genutzt wird. In Swimmer2Race befindet sich die Bahn-Nr. und die amtliche Endzeit dieser Bahn. Ebenfalls sind dort die Zwischenzeiten enthalten. Diese beiden Listen werden nach und nach durchiteriert und dadurch entsteht eine exportierfähige Lenex-Datei.

```
SEND RESULTS;START
<HEAT heatid="3001">
<RESULTS>
<RESULT lane="1" swimtime="00:02:05.00" backuptime1="00:02:05.00" status="">
<SPLITS>
<SPLIT distance="25" swimtime="00:00:30.00" />
<SPLIT distance="50" swimtime="00:01:00.00" />
<SPLIT distance="75" swimtime="00:01:30.00" />
</SPLITS>
</RESULT>
<RESULT lane="3" swimtime="00:03:01.00" backuptime1="00:03:01.00" status="">
<SPLITS>
<SPLIT distance="25" swimtime="00:00:32.00" />
<SPLIT distance="50" swimtime="00:01:04.00" />
<SPLIT distance="75" swimtime="00:01:44.00" />
</SPLITS>
</RESULT>
</RESULTS>
</HEAT>
SEND RESULTS;END
```

Listing 4: *Beispiel einer rückimportierfähigen Lenex-Datei für den Lauf mit der Lauf-ID 3001.*

6.5.3. FileChangeListener (Export der Auswerteergebnisse nach EasyWk)

Der FileChangeListener ist ein von uns entwickeltes Programm, welches weitestgehend die ALGE/EasyWk-Kommunikation simulieren soll. Wie im Entwurfskapitel zum Lenex-Parser beschrieben, erfolgt die Kommunikation mittels splash_receive.txt und splash_send.txt.

Das, was in splash_send.txt gesendet wird, wird verarbeitet und anschließend in splash_receive.txt geschrieben. Der FileChangeListener ist für die Verarbeitung der gesendeten Befehle zuständig. Generell liest das Programm jede 2. Sekunde splash_send.txt ein und bearbeitet jede neue dazugekommene Zeile und bereitet die Antwort für EasyWk vor. Nach erfolgreicher Befehlsverarbeitung aller Zeilen wird splash_send.txt geleert.

Die Methode `interpreteLine(String line)` geht alle möglichen Anfragen durch. Derzeitig wird lediglich überprüft, ob nach Ergebnissen angefragt wurde und ob die ALGE aktiv ist. Anschließend wird aus dem String die HeatID herausgezogen und überprüft, ob die HeatID existiert.

Der FileChangeListener muss die ALGE bzgl. dem `VERSION`-Pinging simulieren. D.h. wenn EasyWk ein VERSION schickt (also VERSION… in der Datei splash_send.txt steht), so muss der Listener ein `VERSION` in splash_receive.txt zurückschreiben.

Der Auswerter-App speichert nach erfolgreichem Lauf die von EasyWk interpretierbare Lenex-Struktur (siehe Unterkapitel Export) in einer Datei <HeatID>.txt. Sofern diese Datei in demselben Ordner des

FileChangeListener befindet, wird diese Datei gelesen und im splash_receive.txt geschrieben. Dort in der Datei enthalten ist bereits der Befehl `SEND RESULTS;START`, welcher dazu dient, dass EasyWk weiß, dass nun eine Abfolge von Lenex-Daten erfolgt und `SEND RESULTS;END` was den Schluss der Übertragung definiert.

Sollte keine Datei <HeatID>.txt existieren, dann wird vom FileChangeListener ein `SEND RESULTS;NOT FOUND` in splash_receive.txt geschrieben, was dazu führt, dass EasyWk keine Antwort erwarten kann, da der Lauf nicht existiert.

6.6 Entwicklungsumgebung und Framework der Kontroll- und Auswerter-Software

In der Planungsphase zu Beginn des Projekts diskutierten die Teams der Auswerter- und Kontrollsoftware den Typ der Anwendung, eine geeignete Programmiersprache und einzusetzende Frameworks. Es wurde als positiv bewertet, dass beide Teams die gleiche Umgebung und gleichen Tools nutzen, um bei etwaigen Problemen besser einander unterstützen zu können. Zuerst wurde überlegt, ob es sich bei dem Programm um eine Desktop- oder einer Web-Anwendung handeln soll. Anforderungen an die Sprache waren, dass sich Test-Driven-Development damit betreiben lässt und im besten Fall sollte es auch eine für die Gruppe bereits bekannte Sprache sein. Außerdem sollte das Erstellen von GUIs mithilfe eines Editors möglich sein, sodass man sich mehr auf die Programmlogik konzentrieren konnten, statt erst alle Facetten der GUI-Programmierung zu lernen. Schließlich hat man sich gegen eine webbasierte Lösung und für eine reine Desktop-Anwendung entschieden. Die Sprache, welche allen am geläufigsten war, ist Java, sodass die Suche nach einem Framework sich schnell auf diese Sprache beschränkte. Außerdem war auch hier die Nutzung von Unit-Tests für TDD bereits bekannt. Als IDE hat man sich für Eclipse entschieden.

Nachdem die Entscheidung also für Java getroffen wurde, wurde nach einem geeigneten Framework gesucht. Primär sollte das Framework das Gestalten von UI-Elementen möglichst einfach machen und gleichzeitig nur einen geringen Einarbeitungsaufwand erfordern.

Swing

Das erste UI-Framework, das man sich angesehen hat, war Swing. Bei Swing handelt es sich um UI-Komponenten, welche alle in der Java-Standardbibliothek enthalten sind. Die Entwicklungsumgebung Eclipse bietet für Swing einen einfach zu benutzenden Editor, welcher GUIs per Drag and Drop gestalten lässt. Die Idee wurde aber schnell verworfen, nachdem klar wurde, dass Swing Komponenten sich schlecht automatisiert testen lassen.

Vaadin

Als nächstes wurde man auf das Framework Vaadin aufmerksam. Positive Aspekte dieses Frameworks sind, dass es sich um ein Web-Framework handelt. Da unser Produkt aus vielen verteilten Systemen besteht, bietet sich eine webbasierte Lösung durchaus an, da sich Kommunikation hier oft einfach lösen lässt. Vaadin legt außerdem besonderes Augenmerk auf das Frontend-Design. Mithilfe eines Eclipse-Plugins gestaltete sich dies als besonders einfach. Ein Nachteil war jedoch, dass für die Nutzung dieses Frameworks eine Lizenz benötigt wird. Als Studenten hatten wir jedoch die Möglichkeit, diese Lizenz umsonst zu bekommen. Der wesentliche Grund, warum wir uns gegen dieses Framework entschieden haben ist allerdings hauptsächlich, dass wir auf JavaFX gestoßen sind und wir gleichzeitig festgestellt haben, dass eine webbasierte Lösung nicht notwendig ist.

JavaFX

Genau wie Swing, ist JavaFX (bis Java 11) Teil der Java-Standardbibliothek. Bis man durch Recherche darauf gestoßen ist, war dies jedoch keinem Gruppenmitglied bekannt. JavaFX versteht sich als Nachfolger von Swing und bietet ein moderneres Aussehen, mehr Funktionalität und bessere Trennung von Design und Logik. Um mit JavaFX schnell und einfach UIs zu bauen, gibt es den s.g. JavaFX Scene Builder. Dieses von Oracle entwickelte Programm erlaubt es UIs per Drag and Drop zu erstellen. Die Designs werden in einer von der Anwendungslogik getrennten Datei gespeichert. Dies hat einen großen Übersichtsvorteil. Mit der Bibliothek TestFX, welche sich leicht mit Maven installieren lässt, ist außerdem eine einfache Gestaltung von automatisierten Tests möglich. Letztendlich haben wir uns für JavaFX entschieden, da es alle unsere Anforderungen erfüllt und außerdem große Importe von externen Bibliotheken nicht notwendig sind.

Zusätzlich zu JavaFX wurde auch das Spring Boot Framework genutzt. Spring Boot ermöglicht Dependency-Injection und verringert den Konfigurationsaufwand eines Maven-Projekts. Spring Boot machte es sehr viel einfacher das Programm in eine ausführbare Datei zu kompilieren, als dies mit einem reinen Maven-Projekt der Fall gewesen wäre.

6.7 Kontrollprogramm

Im folgenden Abschnitt werden die eingesetzten Werkzeuge, die einzelnen Benutzeroberflächen sowie die MQTT-Verbindung des Kontrollprogramms näher erläutert.

6.7.1 Layout und Funktionen

Im Folgenden werden die einzelnen Benutzeroberflächen und deren Funktionalitäten anhand von einzelnen Screenshots vorgestellt. Die dazugehörigen .fxml-Dateien (grafischen Oberflächen) sind unter src/main/resources/view zu finden. Außerdem werden Style-Angaben in einer Datei stylesheet.css definiert, ebenfalls unter src/main/resources/view zu finden.

6.7.1.1 Root (RootLayout.fxml)

Die Oberfläche Root spielt eine zentrale Rolle. Sie wird zu Beginn geladen und gibt die Fenstergröße sowie das Icon vor. Außerdem ist das Root-Layout eine Art Container, bei Aufruf der anderen Oberflächen wird immer wieder die Methode rootLayout.setCenter(view) aufgrufen, die dafür sorgt, dass der Inhalt des Root-Fensters mit dem Inhalt der entsprechenden Oberfläche ausgetauscht wird.

6.7.1.2 Start (Start.fxml)

Zu Beginn des Programms kann zwischen einer neuen Veranstaltung oder einer vorherigen Veranstaltung gewählt werden (Siehe Abbildung 38). Falls eine vorherige Veranstaltung existiert, kann diese aus den Backup-Daten wiederhergestellt werden. Das Backup befindet sich dabei auf der Festplatte und das Programm erkennt diese automatisch. Dies ist gerade bei einen Systemabsturz von großem Nutzen. Weitere Informationen zum Backup unter Punkt 6.7.2.6 Backup.

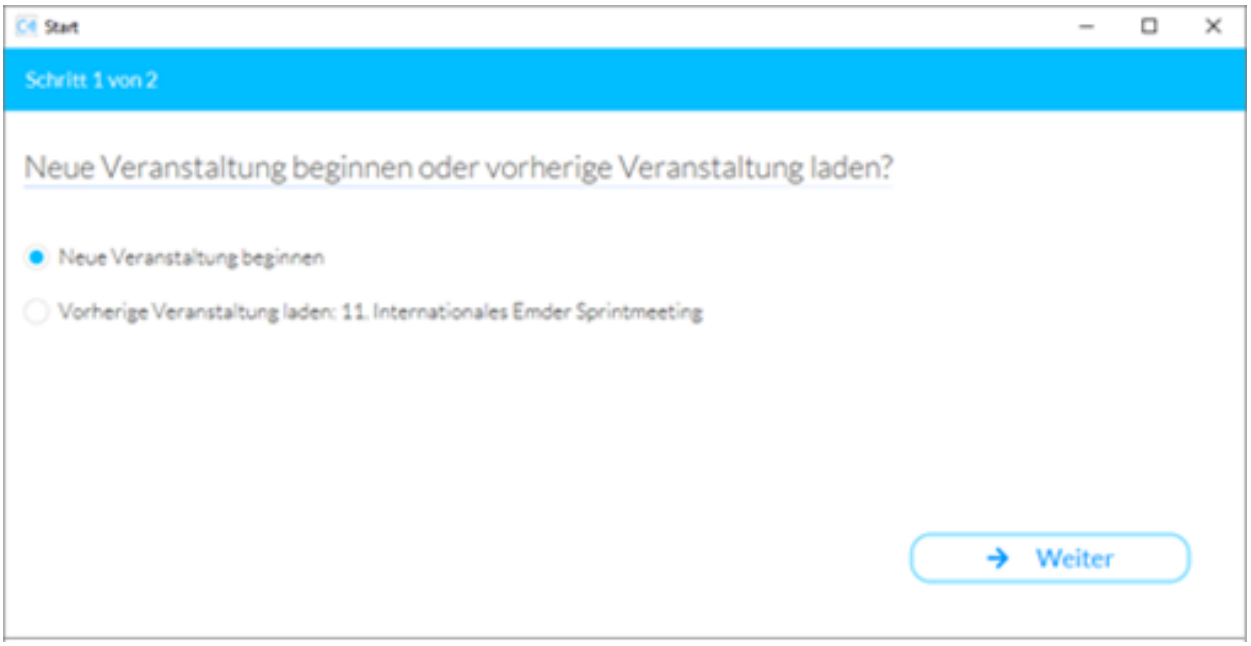


Abbildung 38 Starbildschirm - Schritt 1

Im zweiten Schritt (Siehe Abbildung 39) können die Einstellungen vorgenommen werden. Hier wird u.a. die IP-Adresse des Kontroll-Rechners angezeigt, um eine erfolgreiche MQTT Kommunikation zwischen den Komponenten zu ermöglichen. Die Einstellungen für die Nachlaufzeit (zwischen 0 und 30 Sekunden) eines neuen Startsignals können hier ebenfalls vorgenommen werden. Diese Konfiguration ist gerade bei einem Überkopf-Start wichtig. Nachdem ein Startsignal erfolgt, werden genommene Zeitstempel der Raspberry PIs noch während der Nachlaufzeit dem vorherigen Lauf zugeordnet. Durch den einzustellenden Parameter „Nachlaufzeit bei neuem Start“ wird genau diese Nachlaufzeit festgelegt.

Die Wettkampfdaten müssen im Lenex Format beim Datei Uploader hochgeladen werden.

Bei der Zielrichter-Übersicht können die Token für die einzelnen Zielrichter eingesehen und neu generiert werden. Über den Button „Verbinden“ können die Zielrichter sich mit den Kontrollprogramm verbinden. Dabei gibt der Status in verschieden Farben an, ob der Zielrichter verbunden oder getrennt ist. Die Zielrichter können ebenfalls neu hinzugefügt oder entfernt werden. Bis zu 9 Zielrichter können vorhanden sein.

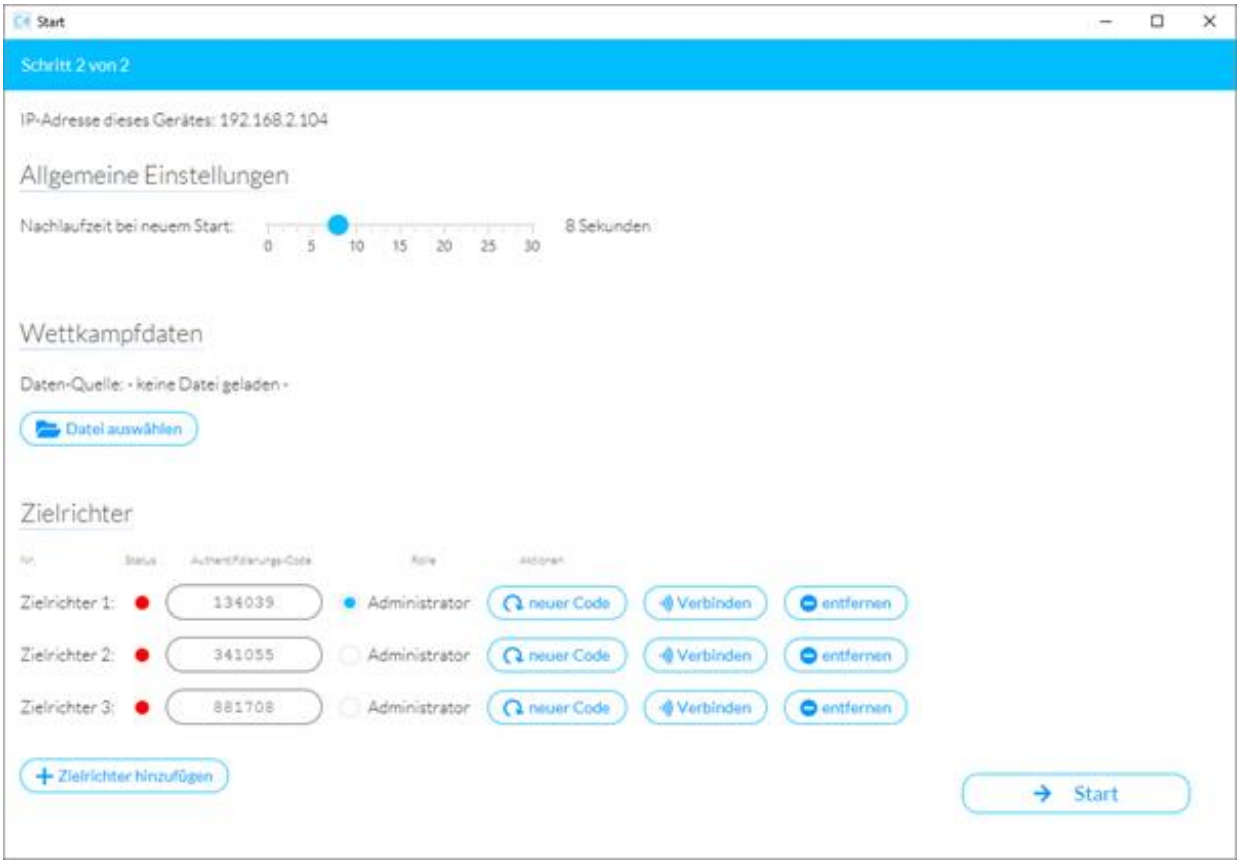


Abbildung 39 Startbildschirm - Schritt 2

6.7.1.3 Dashboard (Dashborad.fxml)

Das Dashboard hat die Anforderungen, alle notwendigen Informationen auf einen Blick übersichtlich darzustellen. Es ist in 3 vertikal getrennten Bereichen aufgeteilt (siehe Abbildung 40).

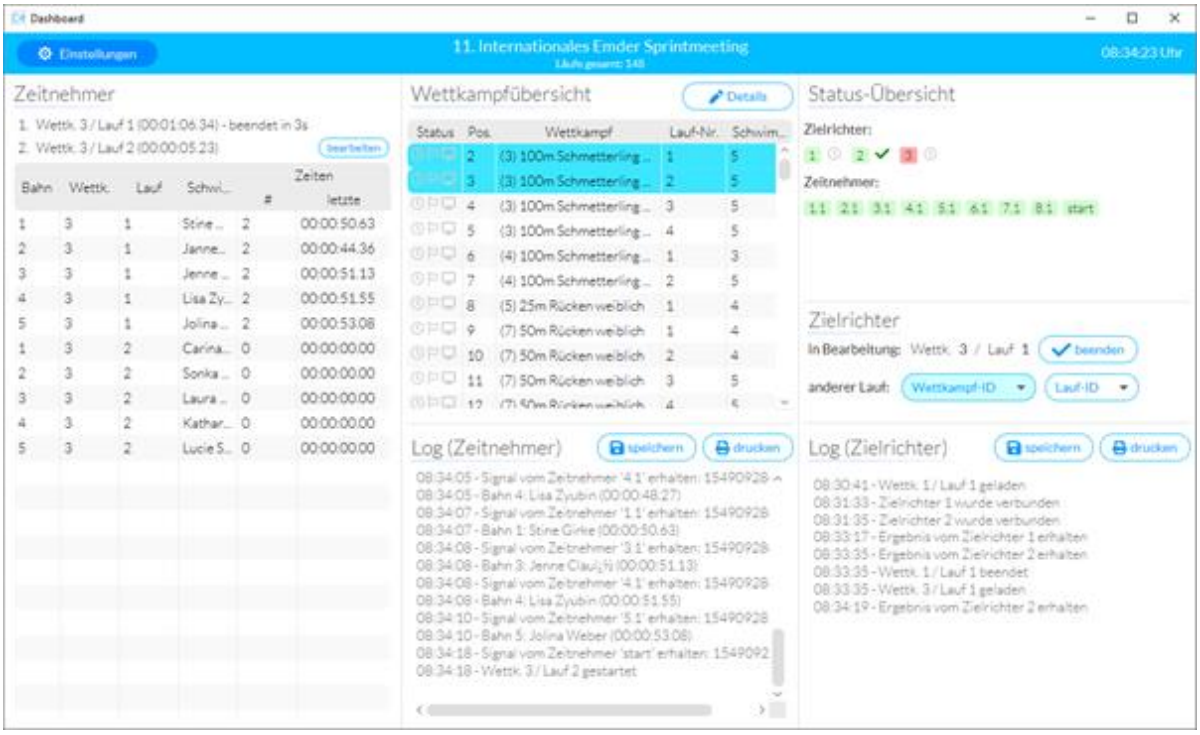


Abbildung 40 Dashboard

Ganz links ist der aktuelle Lauf bzw. die aktuellen Läufe zu sehen. Dort wird angezeigt, welcher Sportler aus welchem Lauf auf welcher Bahn schwimmen. Für jeden Sportler wird noch die Anzahl der bereits genommenen Zeiten und die letzte genommene Zeit angegeben.

Oberhalb dieser Tabelle wird der Lauf bzw. werden die Läufe nochmal mit Lauf-Nr. und Wettkampf-Nr. sowie der laufenden Zeit angezeigt.

Neben diesen Informationen befindet sich ein Button „bearbeiten“, der ein neues Fenster öffnet (siehe Abbildung 41). Dieser ermöglicht dem Benutzer, den Lauf zurückzusetzen (bspw. bei einem Fehlstart oder nach dem Testen), den Lauf zu beenden - damit er dem Auswerter übermittelt werden kann - und er kann den aktuellen Lauf mit einem anderen Lauf tauschen, sodass dieser dann weiterläuft und alle Zeiten der übermittelten Bahnen übernimmt. Diese Funktion ist notwendig, wenn Läufe in der Reihenfolge vertauscht werden oder ein Lauf kurzfristig entfällt und am Kontrollrechner nicht die Zeit blieb, den Lauf in der Wettkampfübersicht zu verschieben oder als beendet zu markieren.

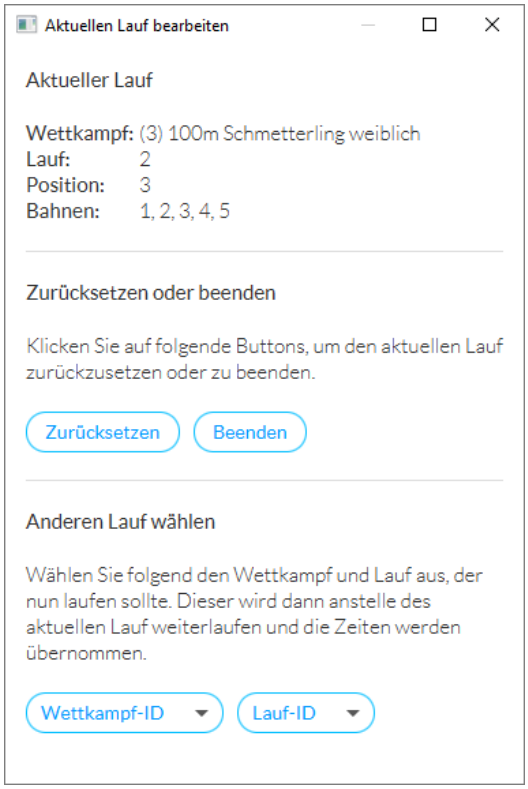


Abbildung 41 Aktuellen Lauf bearbeiten

Der mittlere Bereich (siehe Abbildung 40) ist wiederum horizontal in zwei Bereiche getrennt. Im unteren befindet sich ein Log, in dem alle Ereignisse protokolliert werden, die die Zeitnahme betreffen. Im oberen Bereich wird eine Wettkampfübersicht dargestellt, aus der entnommen werden kann, welche Läufe vorhanden sind – mit Position, Wettkampf-Nr. sowie Beschreibung, Lauf-Nr. und der Anzahl an Schwimmern. Außerdem ist in der Spalte „Status“ zu erkennen, ob dieser Lauf von den Zeitnehmern bearbeitet, den Zielrichtern bearbeitet und dem Auswerter übermittelt wurde.

Der rechte Bereich wird in drei Bereiche getrennt. Im untern befindet sich wieder ein Log, das die Ereignisse der Zielrichter-Apps protokolliert. Sowohl das Log der Zeitnehmer als auch der Zielrichter können jeweils unabhängig gedruckt und gespeichert werden, indem die Buttons oberhalb der Logs verwendet werden. Im mittleren Bereich ist zu erkennen, welcher Lauf bei den Zielrichtern aktuell in Bearbeitung ist. Dieser kann beendet oder ein anderer Lauf kann gewählt werden, der sofort in der App angezeigt wird. Im oberen Bereich befindet sich eine Status-Übersicht, aus der entnommen werden kann, welche Geräte (Zeitnehmer und Zielrichter) verbunden sind und zum anderen, welcher Zielrichter bereits sein Ergebnis abgegeben hat und auf welchen noch gewartet wird.

6.7.1.4 Wettkampfübersicht (ContestEditor.fxml)

Wie bereits beschrieben, verfügt das Dashboard über einen Bereich, in dem die Wettkampübersicht dargestellt wird (siehe Abbildung 43). Nach Klick auf den Button „Details“ wird eine neues Fenster geladen, das die detailliertere Informationen und erweiterte Funktionen zu den einzelnen Läufen bietet. Im linken Bereich wird eine Tabelle geladen, die weitere Informationen zum Lauf gibt. Durch Drag-and-Drop lässt sich ein Lauf in der Reihenfolge verschieben. Durch Klick auf einen Lauf, werden im rechten Bereich weitere Informationen angezeigt.

Dazu gehören die Wettkampf- und Lauf-Nr., der Status der Zeitnehmer und Zielrichtern sowie des Auswerters angezeigt. Falls ein Lauf noch nicht gestartet wurde, wird beim Zeitnehmer ein Button „beenden“ angezeigt, der den Lauf für den Zeitnehmer als beendet markiert, sodass dieser im automatischen Laden des nächsten Wettkampfes nicht beachtet und übersprungen wird. Außerdem wird angezeigt, welche Bahnen durch welchen Schwimmer belegt ist bzw. war und wie die letzte übermittelte Zeit lautet. Diese Anzeige ist hilfreich, wenn man direkt nach einem Lauf die Zeiten wissen und nicht auf den Auswerter warten möchte.

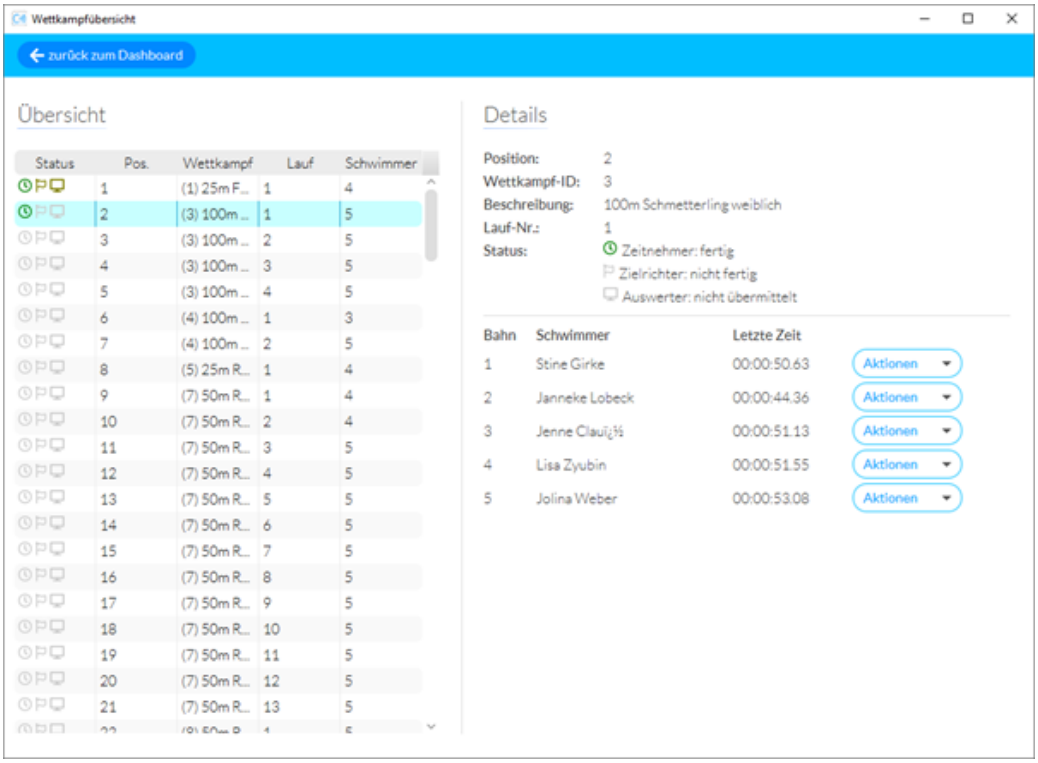


Abbildung 42 Detaillierte Wettkampfübersicht

6.7.1.5 Einstellungen (Settings.fxml)

Die Einstellungen (siehe Abbildung 44) sind ähnlich vom Aufbau wie der Schritt 2 vom Start (siehe Abschnitt 6.7.1.2 Start (Start.fxml)). Änderungen können hier nochmals vorgenommen werden, falls diese beim Start vergessen worden sind. Diese Funktion ist gerade dann von großem Nutzen, wenn z.B. neue Zielrichter während des Wettkampfs hinzugekommen sind, die Verbindung zu den Zielrichtern aus unerklärlichen Gründen getrennt oder die falsche Lenex-Datei eingelesen wurden.

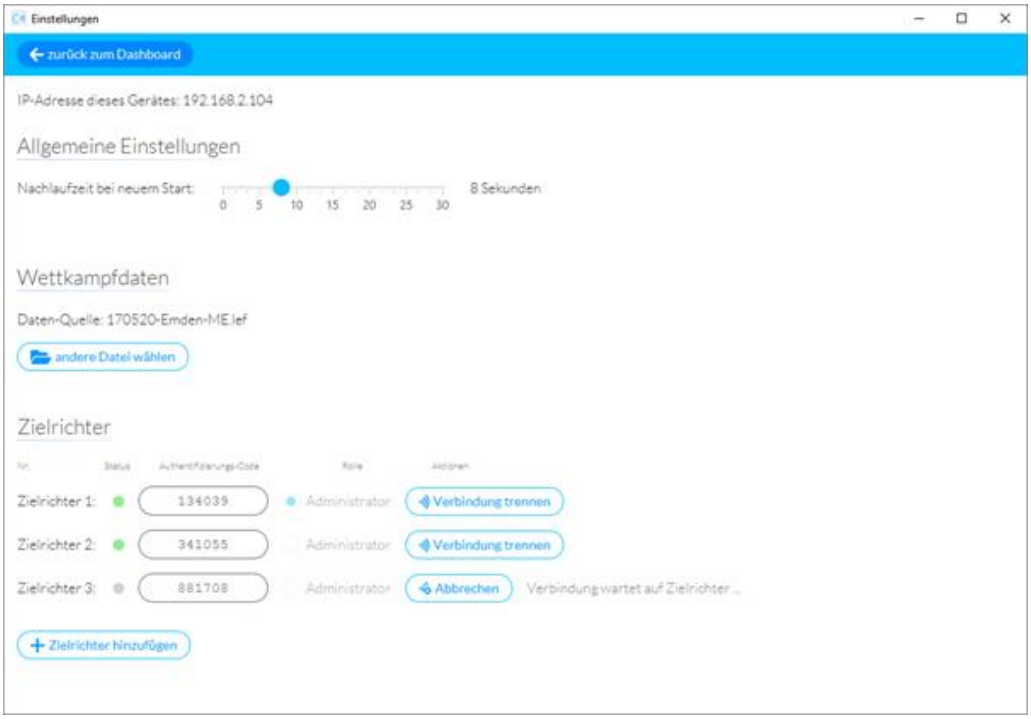


Abbildung 43 Einstellungen

6.7.2 Implementierung

6.7.2.1 Klassen und Controller

Zur besseren Strukturierung wurden die Klassen anhand ihrer Funktion unterschieden und in die Pakete wettkampf, wettkampf.controller und wettkampf.model gespeichert.

Das Hauptpaket wettkampf enthält lediglich die MainApp, den MQTT_Handler sowie das MQTT_Testboard. Die Klasse MainApp startet die grafische Oberfläche, steuert die verschiedenen Ansichten und startet den MQTT-Broker (siehe Abschnitt 6.7.2.2 MQTT). Außerdem sind hier die Methoden zum Speichern und Laden des Backups hinterlegt (weitere Informationen zum Backup im Abschnitt 6.7.2.6 Backup).

Das Paket wettkampf.controller enthält die gesamten Methoden für jede einzelne grafische Oberfläche. Es gibt die Klassen ContestEditorController, DashboardController, SettingsController sowie StartController.

Im Paket wettkampf.model sind nur die Objekte/Modelle abgelegt, die von den Controllern, dem MQTT_Handler oder der MainApp für die Backups verwendet werden.

Folgend werden lediglich besondere Funktionalitäten und Eigenschaften der Implementierungen vorgestellt – die gesamte Implementierung zu beschreiben, ist nicht nötig und wäre zu umfassend. Der Quellcode enthält viele Kommentare, die das Verständnis sehr erleichtern.

6.7.2.2 MQTT

Sowohl der MQTT-Broker als auch der MQTT-Client wurden in der Klasse MQTT_Handler implementiert.

6.7.2.2.1 MQTT-Broker

Für die Implementierung des Brokers wurde die Bibliothek io.moquette in der Version 0.8 genutzt.
Während des Starts des Brokers, wird die Konfiguration des Servers geladen. Die Einstellungen sind in der Datei src/main/resources/config/moquette.conf zu finden. Hier können u.a. der Port sowie die Benutzerauthentifizierung konfiguriert werden.
Sollte der Kontrollrechner die Verbindung mit dem Access-Point verlieren, ist der MQTT-Broker so konfiguriert, dass dieser die Verbindung hält und automatisch wiederherstellt, wenn der Access-Point wieder erreichbar ist.
Da der Kontrollrechner ebenfalls Mitglied im MQTT-Netz zum Datenaustausch ist, zugleich aber auch der Broker, hat man sich entschieden, zwar einen MQTT-Client zum Publishen von Nachrichten anzulegen, nicht aber zum Subscriben. Hierfür wird direkt vom Broker geprüft, um welche Nachricht es sich handelt und die entsprechenden Methoden des Kontrollrechners aufgerufen. In der inneren Klasse MQTT_Listener geschieht dies in der Methoden onPublish(). Diese wird immer aufgerufen, wenn ein Gerät eine Nachricht published. Da alle Geräte nur Nachrichten publishen, die vom Kontrollrechner verarbeitet werden müssen, war es sinnvoll, die Nachrichten bereits in diesem Schritt auszuwerten. Mittels einer Liste “topicList” wurde eine Art Whitelist erstellt, die die topics enthält, auf die der Broker nur reagieren soll. Die Topics sind dem Abschnitt *6.1.2 Kommunikation zwischen den einzelnen Komponenten* zu entnehmen. Sobald eine Verbindung zu einem Zielrichter aufgebaut werden soll, werden entsprechende Topics mit dem Token des Zielrichters ergänzt bzw. beim Abmelden wieder entfernt. Bei den Zeitnehmern wird momentan nur auf die Topics “start” sowie “1.1” bis “8.1” reagiert. Bei weiteren Zeitnehmern müssen diese ebenfalls hinterlegt werden.

Zur Überprüfung der Status der einzelnen Geräte (Zielrichter und Zeitnehmern) hat man mit den Entwicklern vereinbart, dass diese alle 5 Sekunden beliebigen Inhalt auf das topic “status” senden sollen. Nur so kann mithilfe der Timelines-Methoden “checkOnlineState_referees” und “checkOnlineState_timer” überprüft werden, ob die Geräte noch angemeldet sind. Dazu wird der Zeitpunkt des letzten Sendens bei den Zielrichtern im Model Referee und bei den Zeitnehmern in einer HashMap im MQTT_Handler gespeichert. Die Check-Methoden prüfen alle 5 Sekunden, ob das letzte Senden mehr als 10 Sekunden her ist – dann wird das Gerät im Dashboard als offline gekennzeichnet.

6.7.2.2.2 MQTT-Client

Für die Implementierung des Clients wurde die Bibliothek org.eclipse.paho in der Version 1.0.2 genutzt. Zum Publishen wird im MQTT_Handler einer Methode publish(topic, content) bereitgestellt, der man als String das Topic sowie die Nachricht übermittelt. Innerhalb der Methode wird eine neue Verbindung zum Broker mittels Angabe der IP-Adresse und Port aufgebaut und die Nachricht gesendet. Nach erfolgreichem Senden wird der Client wieder getrennt.

6.7.2.2.3 MQTT-Models

Für den strukturierten Austausch zwischen den verschiedenen Teilnehmern, ist es erforderlich, dass man mit dem Partner, der die Daten verarbeitet oder sendet, eine Datenstruktur definiert hat, die von beiden Seiten genutzt werden kann. Die Modelle sind im Paket wettkampf.model zu finden und beginnen mit “MQTT_”.

6.7.2.3 Start und Einstellungen (SettingsController)

Da das Einstellungsfenster dasselbe ist wie zu Beginn im Startvorgang (Schritt 2), sind diese Implementierungen in einem Controller zusammengefasst. Um zu steuern, ob es sich beim Laden des Controllers um das Starfenster oder das reguläre Einstellungsfenster handelt, existiert die Methode setMode(), der man entweder “start” oder “settings” übergeben kann. Dieser Wert wird bei Aufruf von loadContent() berücksichtigt und der Schritt 1 beim Startvorgang vorgeschoben. Außerdem unterscheiden sich die Überschrift und die Buttons ein wenig von dem Einstellungs- und dem Startfenster. Da der gesamte Inhalt abhängig vom Zustand des Programms ist und nur eine grafische Vorlage erstellt wurde, wird der Inhalt dynamisch vom Controller erstellt und ist nicht mit dem SceneBuilder nach Öffnen der Settings.fxml sichtbar. Im SettingsController werden die Zielrichter verwaltet. Hier wird eine ObservableList erstellt, die Zielrichter mittels des Model-Objektes “Referee” enthält.

6.7.2.4 Dashboard (DashboardController)

Der DashboardController bietet die größte Anzahl an Funktionalitäten. Er enthält sämtliche Methoden, um Ergebnisse der Zeitnehmer und Zielrichter entgegenzunehmen und steuert die Anzeigen. Beim Aufruf des Controllers wird in initialize() zuerst geprüft, ob ein Backup geladen werden soll. Falls dies im Startvorgang festgelegt wurde, werden nun die Daten für das Dashboard wiederhergestellt. Außerdem werden andere Vorgänge ausgelöst, u.a. das Starten der Uhrzeit, den Buttons Funktionen zugewiesen, Placeholder erstellt, Sichtbarkeiten einiger Elemente geändert und sämtliche Tabellen mit Werten gefüllt.

Da es bei einem Überkopf-Start für eine gewisse Zeit zwei Läufe gleichzeitig laufen, werden Läufe in einem Array names currentContestsTimer geführt. Um die Logik zu realisieren, dass der Lauf 2 auf den Index von Lauf 1 verschiebt, sobald dieser beendet ist, wurden Hilfsmethoden zum Hinzufügen, Entfernen und Abfragen des Indexes erstellt. Außerdem wird so bei Änderung der aktuellen Läufe eine Methode zum Speichern der Daten im Backup aufgerufen.

Sobald vom MQTT-Broker erkannt wurde, dass eine Zeit vom Zeitnehmer übermittelt wurde, wird im DashboardController die Methode “timerSignal” aufgerufen, dem die ID sowie die übermittelte

Nachricht als String mitgegeben wird. Hier wird nun weiter untersucht, ob es sich um das Signal der Startanlage oder einem Zeitnehmer handelt. Sollte es sich um ein Startsignal handeln, wird die Methode “startRace” aufgrufen, die den nächsten unbearbeiteten Lauf starten lässt. Weiter muss hier untersucht werden, ob es sich um ein Signal für einen Fehlstart handelt. Definiert wurde, dass ein Fehlstart durch mindestens drei Signale innerhalb der ersten 10 Sekunden nach dem Start mitgeteilt wird. Sollte ein Startsignal erkannt worden sein, erscheint ein Popup-Fenster, in dem noch einmal bestätigt werden muss, dass es sich um einen Fehlstart handelt. Nach Bestätigung wird die Methode “startRace” erneut aufgerufen, allerdings mit dem Parameter “0”. Nun werden die Tabellen wieder geleert, Attribute zurückgesetzt und Schwimmerzeiten gelöscht.

Sollte eine Zeit vom Zeitnehmer übermittelt worden sein, so wird anhand der ID die Bahn ausgelesen und die Zeit für bei dem Schwimmer, der aktuell auf dieser Bahn schwimmt, eingetragen. Die Zeiten werden in Millisekunden gespeichert, diese werden vom Auswerter nach der zweiten Nachkommastelle abgeschnitten. Zur Anzeige in der Tabelle werden die Zeiten zu einem String konvertiert. Falls auf der Bahn vom Zeitnehmer aktuell kein Schwimmer vorhanden ist, so wird ein neuer unbekannter Schwimmer auf diese Bahn eingefügt. Für die korrekten Zuordnung dieser Zeit ist der Auswerter zuständig. Der Fehlstart kann auch ohne das Signal der Startanlage ausgelöst werden. Nach Klick auf den Button “bearbeiten” oberhalb der Tabelle der aktuellen Läufe, öffnet sich ein Popup, welches durch die Methode “editRunningRace” beschrieben wird. Weiter ist hier die Möglichkeit gegeben, den aktuellen Lauf gegen einen anderen der Übersicht zu tauschen. Diese Funktion ist nicht in einer eigenen Methode ausgelagert, sondern wird direkt in der o.g. Methode als Funktion in der setOnAction-Methode des “Übernehmen”-Buttons ausgelöst.

Wenn beim MQTT-Broker ein Ergebnis von einem Zielrichter eingeht, wird im DashboardController die Methode refereeSignalResult aufgerufen, die, wie beim Zeitnehmer auch, die ID/den Token sowie die Nachricht als String erwartet. Nun wird sicherheitshalber geprüft, ob das Ergebnis zum aktuellen Lauf gehört und dann beim MQTT-Handler hinterlegt.

Der Lauf gilt als beendet, wenn der nächste Lauf gestartet wurde und die Nachlaufzeit (für den Überkopf-Start) abgelaufen ist und wenn alle Ergebnisse der Zielrichter vorliegen. Wenn alle Zielrichter abgegeben haben, wird ihnen in der Methode “finishRaceReferees” der nächste Lauf zur Bearbeitung bereitgestellt und das MQTT_Evaluate durch den MQTT_Handler abgeschlossen und dem Auswerter übermittelt.

6.7.2.5 Wettkampfübersicht (ContestEditorController)

Dieser Controller steuert die Funktionen der Oberfläche für die Wettkampfübersicht. Dadurch, dass in der Klasse MainApp keine Variable für diesen Controller definiert ist, wird er bei jedem Aufruf neu geladen. Das ist notwendig, damit Änderungen und Aktualisierungen in der Wettkampreihenfolge beachtet werden. So wird bei Aufruf des Controllers durch automatischem Aufruf der Methode “initialize” die gesamte Übersicht aus den Daten Wettkampübersicht des DashboardControllers geladen und es wird in zum aktuellen Lauf gescrollt. Ebenfalls wird in “initialize” das Verhalten für Drag and Drop auf der Liste definiert. Per Drag and Drop wird ermöglicht, die Reihenfolge zu beeinflussen. Nach Klick auf einen Lauf, werden mittels der Methode “loadContestDetails” im rechten Bereich die Details für diesen Lauf geladen.

6.7.2.6 Backup

Backup-Dateien werden im Programmordner mit der Endung .bak angelegt. Im Laufe des Programmverlaufs werden drei .bak-Dateien erstellt.

Zum Speichern der Objekte wurde sich für das JSON-Format entschieden. Dazu wurde die Bibliothek org.hildan.fxgson in der Version 3.1.1 verwendet, die ebenfalls die Properties von JavaFx-Objekten in und aus JSON konvertieren kann.

Eine Datei beinhaltet die allgemeinen Daten zum Event (eventData.bak), wie den Dateinamen, den Veranstaltungsnamen, die Logs von Zeitnehmern und Zielrichtern sowie andere allgemeine Einstellungen und Informationen. Eine weitere Backup-Datei speichert die Lauf-Daten ab (raceData.bak), wie sie auch in der Wettkampfübersicht verwendet werden. In der letzten Datei werden die Zielrichter gesichert (refereeData.bak).

Beim Start wird überprüft, ob bereits ein Backup aus einer vorherigen Veranstaltung besteht, damit diese im Schritt 1 wiederhergestellt werden kann. Die Prüfung ist nur erfolgreich, wenn sowohl die eventData.bak als auch die raceData.bak vorhanden sind und gelesen werden können. Die Daten der Zielrichter werden, ebenso wie die Nachlaufzeit in den Einstellungen, unabhängig von der Entscheidung über Starten einer neuen Veranstaltung oder Wiederherstellen, geladen. So können die Zielrichter die gleichen Tokens wie beim letzten Mal verwenden. Auch die Nachlaufzeit muss nicht jedes Mal angepasst werden, da sich dort in der Regel nach Festlagen dieser nicht viel ändern wird. Sollte das Programm während eines Laufs abstürzen, kann der derzeitige Lauf nach einem Neustart weiter bearbeitet werden. Es fehlen lediglich die Zwischenzeiten, welche geschickt wurden, während das Programm nicht lief. Wird eine Endzeit geschickt, während das Program aus ist, geht dies verloren und müsste im Auswerter-Programm manuell nachgetragen werden.

6.8 Auswertungsprogramm

Das Auswertungsprogramm - oder von uns meist nur schlicht „der Auswerter" genannt - ist für die weitere Verarbeitung bzw. zunächst einmal das Auffangen der vorhandenen Wettkampfdaten zuständig. Nach dem Erhalt der Daten werden diese ohne Umwege im UI des Auswerters dargestellt. Anhand der dort angezeigten Daten soll es jetzt, der für die Auswertung zuständigen Person möglich sein, die Entscheidung der Zielrichter nachzuvollziehen und die Daten auf Vollständigkeit und Widerspruchsfreiheit zu prüfen. Nach der Überprüfung kann der Benutzer den Exportvorgang mit einem Klick auf Speichern auslösen. Somit werden die Wettkampfdaten in das Lenex-Format konvertiert, welches wiederum von EasyWK (dem Schwimmdatenaufzeichnungsprogramm) gelesen werden kann. Als IDE wurde die zu diesem Zeitpunkt neueste Version von Eclipse (Photon Release (4.8.0)) und dazu das Framework JavaFX 2 verwendet. Es wurden dieselben Entwicklerwerkezeuge verwendet, wie für das Kontrollprogramm.

6.8.1 Hauptprogramm (AuswerterAppApplication)

Die Hauptklasse übernimmt einen relativ überschaubaren Teil am Gesamtanteil der Auswerter-App. In JavaFX ist es so gehandhabt, dass die „Main" lediglich die Start-Methode aufruft. Dort wird dann als allererstes die „Stage" gesetzt, die beim Starten der App von der Main als Parameter übergeben wurde. Stage heißt übersetzt so viel wie „Bühne" und ist für JavaFX im Prinzip das Programm. Die Stage kann mehrere Szenen enthalten, die es anschließend darstellen kann. Mithilfe der Stage werden also verschiedene Szenen gesetzt. Diese Szenen sind ganz einfach die unterschiedlichen Fenster, die im übertragenen Sinne die Bühne (Gesamtansicht eines Fensters) mit ihrer Szene (Inhalt eines Fensters/Layouts) besetzen. Im Übrigen übernimmt das Hauptprogramm die Initialisierung der Layouts sowie den Aufruf der zum Hauptlayout dazugehörenden .css-Datei. Es wird außerdem der Konstruktor der Subscriber Klasse mit den hier fest verdrahteten Parameterwerten aufgerufen. Aufgrund dieser Tatsache ist es möglich, dass direkt beim Aufruf der App eine Fehlermeldung empfangen wird, die besagt, dass die Verbindung zum Kontroll-PC fehlgeschlagen ist. Standardmäßig ist der Subscriber zum Testen der App (auch ohne den Kontrollrechner) auf einen öffentlichen „Broker" eingestellt. Diese Einstellung kann im SettingsLayout während der Laufzeit angepasst werden oder grundsätzlich im Quellcode der „AuswerterAppApplication". Obendrein wird beim Start eine Textdatei namens „ContestData" erzeugt, falls diese nicht bereits existiert. Dies wird in der Methode loadContests() bearbeitet. Die Methode erstellt einen neuen FileInputStream, welcher nach der Datei „ContestData" sucht. Existiert diese nicht, wird eine neue, leere Datei erzeugt.

Die Main-App erhält zudem die zuvor vom Kontrollrechner abgesendete, danach von der „messageArrived()" Methode weitergeleitete MQTT-Nachricht. „addMQTTEvaluate()" empfängt also diese Nachricht und formt daraus ein „MQTT-Evaluate-Objekt". Dieses Objekt wird in der Contest-Klasse in ein Contest-Objekt umgewandelt, sodass es hier zur Contest-Liste hinzugefügt werden kann. Anfangs ist die Auswerter-App häufiger abgestürzt. Deswegen gibt es mittlerweile eine Überprüfung, bevor das Objekt auch wirklich zur Contest-Liste hinzugefügt wird. Denn die Kontrollrechner-App sendet bei einer erneuten Anmeldung des Auswerters immer den zuletzt veröffentlichten Wettkampf. Um Duplikate in der Liste zu verhindern, wird die Wettkampf- und Laufnummer des empfangenen Wettkampfes mit denen bereits vorliegenden Wettkämpfen gegengeprüft.

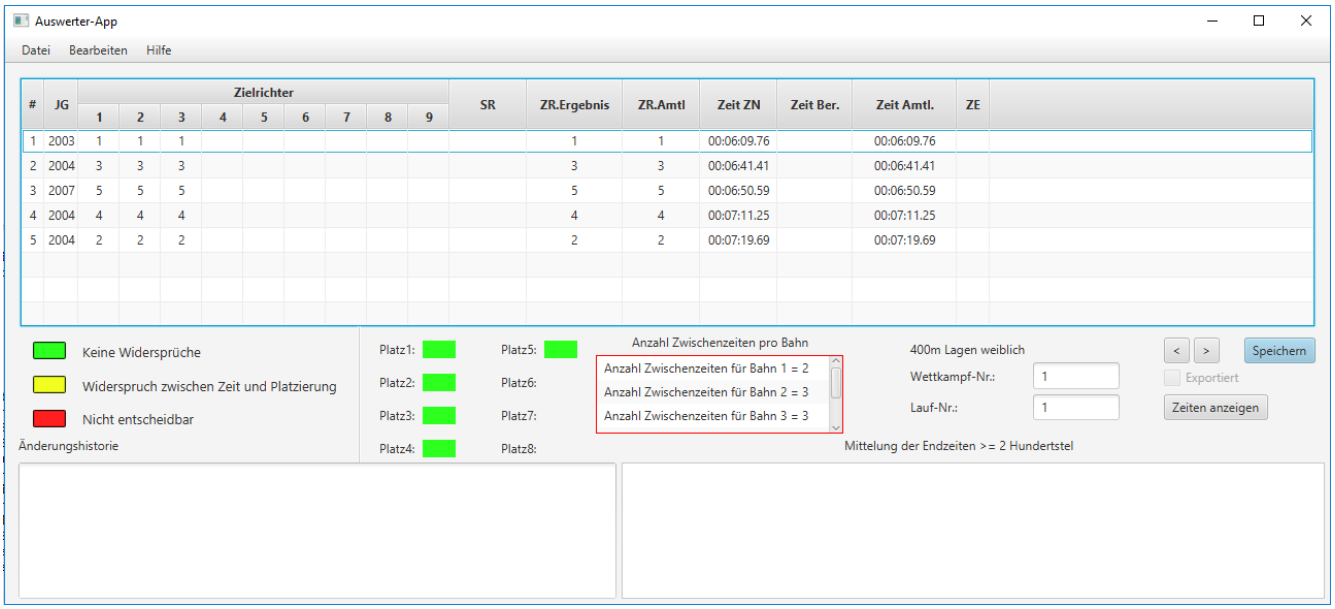


Abbildung 44 Auswertungsprogramm mit beispielhaften Wettkampfdaten

6.8.2 Gestaltungsebenen

Um das Design des Auswerters möglichst modular und strukturiert zu gestalten, wurden für das Hauptfenster zwei Designvorlagen und für jede weitere Ansicht eine neue fxml-Datei angelegt. Mithilfe des "JavaFX Scene Builders" wurden die Benutzerschnittstellen gestaltet und angepasst. Zusätzlich verfügt jedes Design über eine Controller-Klasse, die mit dem UI verknüpft ist und für die Logik der Grafikelemente zuständig ist. Außerdem ist das Hauptlayout noch mit einer .css-Datei (StyleMainLayout) verknüpft. Diese kann genutzt werden, um das Design des Auswerters von einer zentralen Stelle aus zu gestalten (getrennt von der fxml-Datei). Bisher ist dort nur ein Befehl vertreten und dieser dient der Zentrierung aller Werte der Tabelle innerhalb ihrer Zellen.

6.8.2.1 Umrissdesign (Root-Layout)

Mit Umrissdesign ist der äußere Rahmen des Auswerter-Fensters gemeint. Im sogenannten „Root-Layout wird also die Breite und Höhe des Fensters definiert. Dazu zählen auch die standardmäßigen Reiter: Datei, Bearbeiten und Hilfe sowie die rechts oben verfügbaren Optionen zum Anpassen der Fenstergröße bzw. Schließen des Fensters. Bevor das Programm endgültig geschlossen werden kann, kommt immer zunächst eine Abfrage, die es verhindern soll, das Programm versehentlich zu beenden. Die Standard-Reiter haben in dieser Anwendung bisher kaum eine Funktionalität. Unter Bearbeiten lassen sich die Einstellungen aufrufen. Deshalb dienen die Reiter bisher hauptsächlich der Usability bzw. sollen eine gewohnte Umgebung für den User schaffen.

6.8.2.2 Hauptansicht (Main-Layout)

Wie in Abbildung 44 zu sehen besteht der hauptsächliche Teil aus einer Tabellenansicht. Die Tabelle besteht aus 8 Zeilen, jeweils für maximal acht Bahnen bzw. Schwimmteilnehmer und vielen verschiedenen Spalten. Es beginnt mit den Positionen, die immer aufsteigend sortiert sind. Die letzte/höchste Ziffer gibt im Normalfall gleichzeitig an wie viele Bahnen/Schwimmer in diesem Wettbewerb belegt/gemeldet sind. Darauf folgt die Jahrgangsspalte. Es handelt sich hierbei lediglich um eine Information, die mit aufgenommen wird. Allerdings kann diese Spalte bei Unstimmigkeiten in der Platzierung helfen, denn ein unterschiedlicher Jahrgang macht es unter Umständen obsolet einen eindeutigen/endgültigen Sieger zu bestimmen. Für die Zielrichterentscheidungen sind immer fest 9 Spalten vorhanden. Damit kann es maximal 9 Zielrichter geben. In der Zielrichterspalte stehen die Bahnen für den jeweiligen Platz. In diesem Beispiel waren sich die Zielrichter immer alle einig und haben Bahn 1 auf Platz 1, Bahn 3 auf Platz 2 gesehen usw. Die SR/Schiedsrichterspalte ist eine Information, die zu Dokumentationszwecken dienen soll. Das Zielrichterergebnis setzt sich aus den Bewertungen der Zielrichter zusammen und bildet damit den Mehrheitsentscheid ab. Hier kann es auch durchaus vorkommen, dass anhand der Zielrichterergebnisse keine eindeutige Platzierung ausgelotet werden kann und das Feld daraufhin mit n.E. (nicht entscheidbar) gekennzeichnet wird. Dadurch, dass auch mehrere Bahnen oder auch keine in der Zielrichterspalte angegeben werden können, ist die Entscheidung in der ZR.Ergebnis-Spalte nicht trivial und wird unter Punkt 6.8.3.2 detaillierter beschrieben. Die ZR.Amtl Spalte beinhaltet das amtliche Ergebnis. Grundsätzlich wird an dieser Stelle das Ergebnis der ZR.Ergebnis-Spalte übernommen. Dennoch handelt es sich hierbei um eine bearbeitbare Spalte. Es können also manuelle Änderungen am Ergebnis vorgenommen werden. Dieser Punkt ist ein wichtiger Punkt der Requirements. Die im Kampfgericht zuständige Person für die Auswertung wird durch das Auswerterprogramm bei der Bestimmung der amtlichen Platzierung und der amtlichen Zeiten unterstützt – diese Person ist aber letztlich verantwortlich und muss in der Lage sein, den amtlichen Zielrichterentscheid und die amtlichen Zeiten manuell festzulegen. Des Weiteren ist es denkbar, dass ein Schwimmer aus irgendwelchen Gründen aus der Wertung entfernt wird. Möchte man eine Platzierung komplett aus der Wertung entferne, so ist dies mit der Eintragung einer „0" in der Spalte „ZR.amtl" erreichbar. Dadurch wird dann ein „n.E." eingetragen und die Zeiten entfernt. Ein regulärer Ausdruck kümmert sich dabei darum, dass der eingetragene Wert zulässig ist (0, 1-8). Die Zeit ZN ist die vom Zeitnehmer erhaltene Zeit. Zeit Ber. wird automatisch vom Programm berechnet, wenn es eine Diskrepanz zwischen der Zeit und der Platzierung gibt. Zeit amtl. übernimmt die Zeiten aus Zeit ZN oder wenn vorhanden aus Zeit Ber.. Das ZE oder auch Zielrichterentscheid Feld bleibt meistens leer, jedoch dient es in kritischen Fällen dazu, zu erfassen, wie die letztendliche Entscheidung zustande gekommen ist.

Kommen wir nun zu den Funktionalitäten und Grafikelementen unterhalb der Daten-Erfassungstabelle. Links am Rand befinden sich drei verschiedene Zustände, die farblich zugeordnet werden. Es handelt sich dabei um eine Art Legende. Bei Grün ist alles in Ordnung. Gelb zeigt an, dass die Platzierungsreihenfolge der Zielrichter nicht mit der zeitbasierten Platzierung übereinstimmt. Als Beispiel: Der Schwimmer auf Bahn 1 hat eine Zeit von 6:00,00 Min und wird von den Zielrichtern auf Platz 3 gesehen. Der Schwimmer auf Bahn 4 hat eine Zeit von 5:59,90 Min und wird auf Platz 4 gesetzt. Da die Zielrichterentscheidungen immer bevorzugt werden, kann es sein, dass dieses Ergebnis vom Auswerter genauso übernommen wird. Jedoch ist es wichtig solche Vorfälle zu markieren und darauf aufmerksam zu machen, bei großen Diskrepanzen müssen eventuell weitere Informationsquellen hinzugezogen werden. Rot zeigt dagegen an, dass die Zielrichter sich nicht einig sind und kein Mehrheitsentscheid getroffen werden kann.

Diese farbigen Markierungen werden rechts daneben, den Positionen (Plätzen) zugeordnet. Wenn hier alles grün ist, ist die Wahrscheinlichkeit sehr hoch, dass es keine Probleme bei der Datenerfassung gab und somit kann anhand dieser Visualisierung entsprechend schneller zum nächsten Wettkampf gewechselt werden.

Der kleine Kasten daneben zeigt an, mit wie vielen und ob alle Bahnen mit gleich vielen Zwischenzeiten bestückt sind. Sobald mindestens eine Zwischenzeitenanzahl abweicht, erscheint eine rote Umrandung des Kastens, die genau diesen Umstand visualisieren soll (für weitere Details siehe Abschnitt *6.8.2.4 Zeitansicht (Time-Layout)*) .

Im Bild wird darauffolgend die Namensbezeichnung des aktuellen Wettkampfes dargestellt. Zusätzlich ist es darunter möglich, durch eine manuelle Eingabe der Wettkampf- und Laufnummer einen bestimmten Wettkampf aufzurufen, sofern dieser bereits vorhanden ist. Falls dies nicht der Fall sein sollte, wird dies durch ein entsprechendes Hinweisfenster symbolisiert. Führende Nullen in der Eingabe werden automatisch rausgekürzt. Es bietet sich ebenfalls die Möglichkeit, die zwei Pfeiltasten für die Wettkampfnavigation zu verwenden. Wenn kein vorheriger oder nächster Wettkampf vorhanden ist, wird die entsprechende Pfeiltaste automatisch ausgegraut bzw. deaktiviert, um unangenehme Pop-up-Fenster zu vermeiden. Ein Klick auf den „Zeiten anzeigen" Button öffnet das Time-Layout-Fenster und wird in Abschnitt 6.8.2.4 genauer beschrieben. Wie zuvor erwähnt, gibt die rote Umrandung des Kastens mit den Zwischenzeiten Aufschluss darüber, dass es hier einen Fehler gibt, was zumeist der Auslöser dafür ist, sich die einzelnen Zeiten im Time-Layout etwas genauer anzusehen. Der „Speichern-Button" löst die Konvertierung der aktuellen Wettkampfdaten in das für EasyWK lesbare Lenex-Format aus. Nach der Betätigung des Buttons ist ein Haken in der für den User selbst deaktivierten „Exportiert-CheckBox" zu erkennen. Anhand dieses Feldes ist es für den Nutzer einfach nachzuvollziehen, ob der Exportiervorgang bereits stattgefunden hat.

Die Änderungshistorie links unten wird automatisch mit getätigten Anpassungen pro Wettkampf befüllt. Heißt, wenn eine Zeit geändert, gelöscht oder ein ZR.Ergebnis geändert wird, erscheint in der Änderungshistorie eine Zeile, die die getätigte Änderung dort dokumentiert. Für jeden Wettkampflauf existiert eine eigene Änderungshistorie. Auf diese Weise können die manuell von der Auswerterperson getätigten Schritte nachvollzogen und auch wieder rückgängig gemacht werden, da hier sowohl die vorherigen als auch die neuen Werte aufgeführt werden. Die Mittelung der Endzeiten im weißen Kasten daneben führt hingegen alle Bahnen auf, deren gestoppte Zeit und dazugehörige berechnete Zeit sich um mindestens 20 Hundertstel unterscheiden. Ein Beispiel für eine solche Mittelung: Der Drittplatzierte hat eine Zeit von 22 Minuten und der 2. Platzierte eine Zeit von 23 Minuten. Da es nicht sein kann, dass der Dritte eine bessere Zeit als der Zweite hat, wird hier gemittelt und in diesem vereinfachten - aber realitätsfernen - Beispiel hätte nun jeder eine Zeit von 22:30min und würde in dem Feld angezeigt werden. Die Mittelung von Zeiten kann übrigens auch über mehrere Bahnen hinweg erfolgen. Grundsätzlich ist es im Schwimmwettkampf wichtig zu erkennen, wenn um mehr als 20 Hundertstel gemittelt wurde, was in diesem Beispiel klar der Fall ist. In diesem Fall muss gemäß der Wettkampfbestimmungen des Deutschen Schwimmverbandes (siehe §134 Abs. 3.d in [19]) der Schiedsrichter die amtlichen Zeiten festlegen.

Das Mainlayout enthält außerdem auch noch Funktionalität, um dem Benutzer mitzuteilen, dass ein neuer Wettkampf zur Auswertung angekommen ist. Da ein Aktionsfenster in der Mitte der Anwendung stören würde und diese außerdem aktiv vom Benutzer geschlossen werden muss, hat man sich für ein kleines Infofenster am unteren rechten Bildschirmrand entschieden. JavaFX enthält standardmäßig kein solches Fenster, weswegen zusätzlich die Bibliothek „ControlsFX" (http://fxexperience.com/controlsfx/features/) per Maven eingebunden wurde.

6.8.2.3 MQTT-Einstellungsfenster (Settings-Layout)

Die MQTT-Einstellungen sind in einem kleinen schlicht gehaltenen Fenster enthalten (siehe Abbildung 46), welches sich über die Auswahl „Bearbeiten/Einstellungen" aufrufen lässt. Die Einstellungen werden zumeist beim Start der Anwendung benötigt, um dort kurz die aktuellen Daten anzugeben. Der MQTT-Broker ist dabei der Kontrollrechner bzw. die IP des Kontrollrechners, der die Daten an den Auswerter weiterleitet. Die Einstellungen verfügen ebenfalls über eine Kontroller-Klasse. Diese beinhaltet die Funktionen „handleApply()" und „handleCancel()". Zweitere ist trivialer Natur und schließt lediglich das Fenster. Bei der ersten Methode werden die drei Felder auf ihren Inhalt geprüft. Wenn mindestens ein Feld leer ist, führt dies bereits zu einer Fehlermeldung, die dazu auffordert, alle Felder auszufüllen. Sofern alle Felder einen Wert aufweisen, folgen weitere Validierungen. Hat dies soweit funktioniert, wird versucht eine MQTT-Verbindung aufzubauen. Gelingt dieser Verbindungsaufbau, wird das Einstellungsfenster geschlossen. Ist dies nicht der Fall, gibt es eine Fehlermeldung und die Eingabedaten werden wieder mit den ursprünglichen Informationen gesetzt. Alle MQTT-Funktionalitäten befinden sich in der Subscriber-Klasse. Die MQTT-Evaluate-Klasse ist hingegen eine Schnittstelle, auf die sich geeinigt wurde und Felder für den Informationsaustausch enthält.

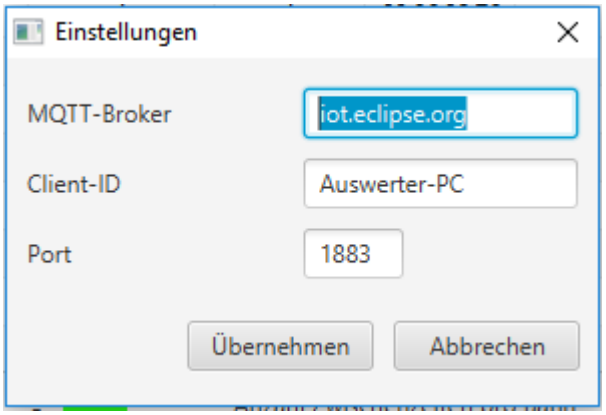


Abbildung 45 MQTT Einstellungsfenster

6.8.2.4 Zeitansicht (Time-Layout)

In der Hauptansicht des Auswerters wird lediglich die Endzeit dargestellt. Dennoch gibt es einige Schwimmwettkämpfe, die sehr viele Zwischenzeiten beinhalten. Zum Beispiel gibt es bei einer Strecke von 1500m insgesamt 14 Zwischenzeiten und eine Endzeit. Im Wesentlichen ist es korrekt pro 100m eine Zwischenzeit zu erwarten. Deshalb wurden hier auch für jede Zwischenzeit immer hartkodiert 100m Schritte eingestellt. In der Zeitansicht hat man eine gute Übersicht über alle übermittelten Zeiten. Durch einen Klick auf den Dropdown-Button für die jeweilige Zeit werden alle vorhandenen Zwischenzeiten und am Ende die Endzeit angezeigt, sodass die Zeiten aufsteigend sortiert zu sehen sind. In diesem Beispiel (siehe Abbildung 46) handelt es sich um eine 400m lange Strecke. Es werden dementsprechend 3 Zwischenzeiten erwartet. Bahn 1 hat lediglich 2 Zwischenzeiten und die 1. Zeit ist fast doppelt so lang. Außerdem lässt die Endzeit von Bahn 1 darauf schließen, dass derjenige schneller als alle anderen war. Daraus ließe sich ableiten, dass in Wirklichkeit mit hoher Wahrscheinlichkeit die 1. Zwischenzeit fehlt und die 2. Zwischenzeit der Zeit der 1. Zwischenzeit entspricht. Bisher ist es jedoch nur möglich Zwischenzeiten zu löschen, indem die gewünschte Zeit mit dem Dropdown-Button ausgewählt und danach der Löschen-Button betätigt wird. An dieser Stelle kann der hier aufgetretene Fall noch nicht gelöst werden. Hierfür müsste es möglich sein, Zwischenzeiten hinzuzufügen und die Zeiten an sich zu ändern. Eine solche Anpassung muss momentan noch manuell in EasyWK geschehen.

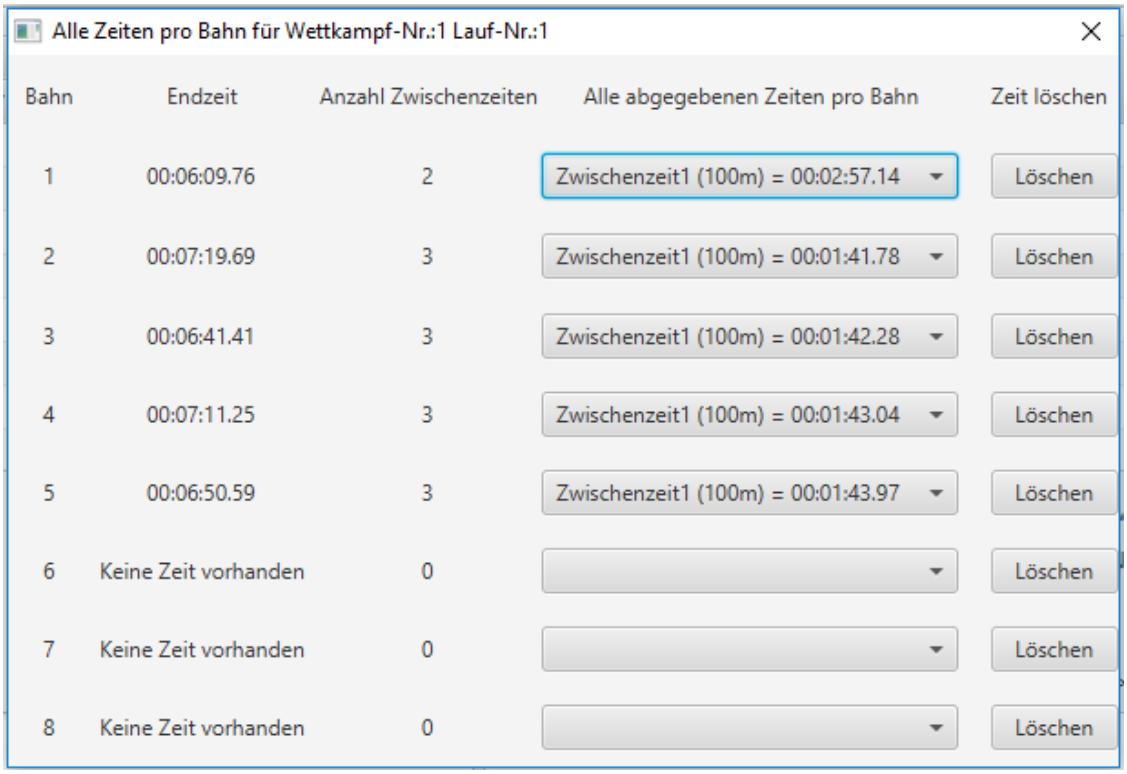


Abbildung 46 Zeitansicht mit beispielhaften Wettkampfdaten

6.8.2.5 Zwischenspeichern der Wettkämpfe

Um Wettkämpfe persistent speichern zu können, sodass nicht alle Daten verloren gehen, sobald das Programm geschlossen wird, müssen diese auf der Festplatte abgespeichert werden. Jedes Mal, wenn ein Wettkampf empfangen wird, wird geprüft, ob dieser Wettkampf bereits in der internen Wettkampfliste vorhanden ist. Wenn ja, wird dieser Wettkampf übersprungen. Falls nicht, wird er der Liste hinzugefügt und die gesamte Liste daraufhin abgespeichert. Zum Speichern wird die Methode saveContests() aufgerufen. Ursprünglich sollte die Liste mit allen Contest-Objekten einfach serialisiert und abgespeichert werden. Dies funktionierte allerdings nicht, da sämtliche JavaFX-Klassen nicht das Interface „Serializable“ implementieren. Als Workaround wird einfach die Liste als String konvertiert und der dabei entstehende String serialisiert und abgespeichert. Um einen String zu erstellen, welcher nachträglich wieder in eine Liste umgewandelt werden kann, muss eine Bibliothek wie Gson verwendet werden. Gson kann allerdings keine JavaFX Objekte konvertieren, sodass hier die zusätzliche Bibliothek FxGson ([13]) verwendet wurde, welche genau dieses Problem adressiert.

6.8.3 Klassen und Logik

Die Contest- und Placement-Klasse sind die Hauptklassen für die Erfassung und Aufnahme der Daten zur Laufzeit. Des Weiteren existieren die Klassen: „MQTT_Evaluate”, „Result” und „ZielrichterResult”. Sie dienen zum Auffangen der vom Kontrollrechner bereitgestellten Informationen und die Attribute wurden im Vorfeld fest definiert. Ein Placement-Objekt hingegen enthält alle in der Tabelle aufgeführten Informationen. Die Contest-Klasse besteht aus einem Placement-Objekt und zusätzlich noch aus den weiteren Daten, die in der Hauptansicht zu sehen sind. Ein Contest-Objekt verfügt also nicht nur über die Platzierungsinformationen, sondern auch noch über wettkampfspezifische Infos. Bei der Betrachtung der Attribute fällt auf, dass die meisten Variablen vom Typ „StringProperty” sind. Bei der StringProperty handelt es sich um eine abstrakte Wrapper-Klasse. Eine mögliche Implementierung davon ist die „SimpleStringProperty”. Dieser Datentyp wird benötigt, um die Variablen an Textfelder und andere Oberflächenelemente zu binden. Diese Bindung geschieht in der jeweiligen Kontroller Klasse.

6.8.3.1 Contest Klasse

Die Klasse besteht aus zwölf privat gekapselten Attributen mit jeweils eigenen trivialen „Setter- und Getter“ Methoden, um die Attribute zu setzen und abzurufen. Zum Erstellen eines Objekts existiert ein parameterloser Standard-Konstruktor, welcher nur zu Testzwecken dient. Der Standard-Konstruktor initialisiert nur alle Attribute. Ein weiterer Konstruktor nimmt Parameter für alle Attribute an und setzt diese. Danach werden alle Platzierungen erstellt und mithilfe des Zielrichter-Algorithmus die Platzierung bestimmt. Nach der Platzierung werden zum Schluss noch die Zwischenzeiten und die Endzeit auf die jeweilige Bahn gemapped. Nachdem mit der Kontrollrechner Gruppe eine Klasse für den Datenaustausch erstellt wurde, musste aus den übertragenen Daten ein Contest-Objekt erstellt werden können. Aus diesem Grund wurde noch eine Factory-Methode geschrieben. Diese Methode erwartet ein MQTT_Evaluate-Objekt als Parameter (also ein Objekt aus dem Datenaustausch) und konvertiert alle Daten in das benötigte Format. Nach der Konvertierung wird der oben beschriebene Konstruktor mit allen Parametern aufgerufen.

Außerdem existieren noch einige Methoden, welche innerhalb des Konstruktors aufgerufen werden. **calcTimes():** Prüft für den gesamten Contest, ob sich Zeiten und Platzierungen widersprechen und mittelt die Zeiten gegebenenfalls.

fixZielrichterResults(): Diese Methode schiebt die Platzierungen, die von den Zielrichtern eingegeben wurde so rum, dass Freiräume nur dort entstehen, wo sie hingehören. Hat ein Zielrichter beispielsweise Position 1 ausgelassen, jedoch Position 2 und weiter mit Bahnen belegt, würden alle einen nach oben geschoben werden. Für den Fall, dass ein Zielrichter zwei Bahnen auf eine Position gesetzt hat, muss die Position darunter frei bleiben. Sind zwei Bahnen auf Position 1 gesetzt worden, kann keine dritte Bahn den zweiten Platz innehaben. Auch dieser Fall wird beachtet und die Bahnen entsprechend geschoben. Diese Methode wurde jedoch in letzter Minute aus dem finalen Produkt entfernt, da laut eines Offiziellen des Schwimmvereins diese Funktionalität nicht gebraucht wird. Die Methode ist noch im Code erhalten, wird aber nie aufgerufen.

6.8.3.2 Placement Klasse

Der Grundaufbau der Klasse besteht aus verschiedenen Attributen, die zur Speicherung der Platzierungsinformationen gebraucht werden. Die Attribute in der Placement-Klasse sind identisch mit den Spaltennamen der Tabelle in der Hauptansicht. Hauptsächlich findet in dieser Klasse der Zielrichter-Algorithmus in verschiedenen Methoden statt. Es gibt einige Hilfsfunktionen wie z.B. „timeCalcString()” und „timeCalcLong()”. Diese zwei Methoden werden mehrfach auch in anderen Klassen verwendet, um die Zeit in ein lesbares Format zu konvertieren (timeCalcString()) und wieder in Millisekunden umzuwandeln (timeCalcLong()). Die Funktion „calcLaneForPosition()” nimmt als Parameter eine Liste von Zielrichterentscheiden auf. Diese Zielrichterentscheide werden pro Position angeordnet und danach wird überprüft wie oft eine Bahn an dieser Position vorkommt. Die am häufigsten auftretende Bahnziffer an der Position wird zurückgegeben. Sollte es zwei oder mehr Elemente mit einer identischen Häufigkeit geben, wird stattdessen „-1” zurückgegeben und es handelt sich um ein nicht entscheidbares Ergebnis. Die Funktion „findBiggestValue()” dient dabei speziell als Hilfsfunktion um die Häufigkeiten gegeneinander abzuwägen. Dann gibt es noch „setRefereesForPosProperty()” sowie „getRefereesForPos()”. Erstere benutzt die zweite Funktion zunächst als Hilfsfunktion. Denn getRefereesForPos sucht die Bahnen eines Zielrichters für genau eine Position heraus. D.h. setRefereesForPos wird pro Zielrichter aufgerufen, während die get-Funktion von der set-Funktion pro Position aufgerufen wird. Nachdem die Hauptfunktion dann die Ergebnisse pro Zielrichter pro Bahn hat, werden diese Bahnen per Komma separiert und in eine Form gebracht um in der Datenerfassungstabelle dargestellt werden zu können. Da es manchmal vorkommen kann, dass Schwimmer nicht antreten, gibt es z.B. die „assignAge” Methode, die dafür sorgt, das Alter/Jahrgang

auszublenden, wenn es keine Ergebnisse für diese Bahn gibt. “fillTimeOfficial()” ist dafür zuständig die berechnete Zeit in die amtliche Spalte einzutragen. Falls es eine gemittelte/berechnete Zeit für eine Position gibt, wird immer diese verwendet, sollte dort kein Eintrag existieren wird die vom Zeitnehmer gestoppte Zeit verwendet.

6.8.4 Probleme

Während der Implementierung sind des Öfteren Probleme aufgetreten. Manche wurden direkt als solche erkannt. Andere entpuppten sich erst später als solche, z.B. durch ungenau definierte Anforderungen. Die Anforderungen wurden zunächst nach dem aktuellen Wissensstand bearbeitet, manche stellten sich jedoch erst später als falsch heraus. Dazu kamen noch Schwierigkeiten mit dem Framework Java FX, das in diesem Projekt – für die Projektgruppe – das erste Mal zur Anwendung kam.

6.8.4.1 Struktur der Auswerter App/Zuordnung zwischen Bahnen und Positionen

In der zuerst entworfenen Tabelle für die Aufnahme der Schwimmdaten, befand man eine Zuordnung der Bahnen zu Platzierungen als äußerst sinnvoll. Denn eine Bahn kann nur eine Platzierung umfassen. Eine Platzierung kann aber mehrere Bahnen haben (Klassische 1:n Beziehung). Demnach kommt es bei der Zuordnung pro Platz sehr oft vor, dass die Zielrichter in eine Zelle mehrere Bahnen eintragen. Logischerweise ist dieses Konzept sehr viel komplizierter auszuwerten, als immer nur einen Zahlenwert in einer Zelle vorzufinden. Denn es ist nie klar, wie viele Werte denn jetzt in einer Zelle stehen. Die Anzahl liegt somit zwischen 0 bis Anzahl-Bahnen. Da es sich hierbei nichtsdestotrotz um ein bereits in dieser Weise handschriftlich angewandtes Modell handelte, gab es keinen Spielraum für Diskussionen. Die Zielrichter würden also kein anderes/neues Modell zur Erfassung akzeptieren. Dementsprechend wurde das Modell auf die Bahnen pro Platz ummodelliert. Im Gegensatz dazu werden intern zur Verarbeitung Datenstrukturen wie HashMaps und Listen verwendet, innerhalb dieser, die Daten aufbereitet und in ein auswertbares Muster umgewandelt werden.

6.8.4.2 Zellen farbig markieren

Zu Beginn war es Teil der Anforderungen, anstatt der jetzt existierenden Abbildung der Plätze mit den dazugehörigen Farben, die Zellen direkt zu highlighten, z.B. in der Spalte ZR.Ergebnis. Diese Anforderung hat sich jedoch als aufwändiger als gedacht herausgestellt. Um die Daten innerhalb der Tabellen zu setzen, werden sogenannte „cell value factories” verwendet. Wenn man allerdings die Farbe, der Zellen ändern will, überschreibt man damit die Werte. D.h. es geht nur eins von Beidem. Alternativ wäre auch beides gleichzeitig möglich, wenn man dafür eine neue Klasse erstellt, die nur für das Highlighten zuständig ist und eine neue zweite „cell value factory“ implementiert. Aufgrund von Zeitmangel ist es dadurch nötig geworden einen schneller umsetzbaren “Workaround” zu finden. In diesem Fall wurde die Markierung stattdessen von den Zellen getrennt, was wesentlich schneller realisierbar war und eine ähnliche Funktionalität bietet.

6.8.4.3 Algorithmus für die Zwischenzeiten

Kurz vor der Feuerprobe unseres verteilten Softwaresystems wurde nochmal darauf aufmerksam gemacht, dass bei den bevorstehenden Wettkämpfen ein großes Augenmerk auf den Zwischenzeiten liegen wird. Die Zwischenzeiten wurden zuvor gänzlich ignoriert, da die Endzeit, die eigentlich entscheidende Rolle einnimmt. Das heißt die Zwischenzeiten wurden zwar abgespeichert, jedoch noch nicht dargestellt. Es wurde immer angenommen, dass die zuletzt gestoppte Zeit die Endzeit ist. Dennoch bestand der bevorstehende Wettkampf aus vielen sehr langen Strecken und im 1500m Freistil-Wettkampf liegen sogar 14 Zwischenzeiten vor. Zwischenzeiten sind immer sogenannte Split-Zeiten und keine Rundenzeiten (kumuliert). Wenn der Sportler z.B. bei 500m angekommen ist, entspricht die Zwischenzeit der Zeit, die er für die kompletten 500m benötigt hat und nicht jeweils der

Rundenzeiten für die einzelnen 100m Strecke. Die Werte der Zwischenzeiten sind also immer aufsteigend. Beim Erstellen eines Algorithmus für diese Zeiten treten allerdings viele kleine Probleme auf. Zum einen standen uns am Anfang keine Informationen darüber zur Verfügung wie viele Meter der Wettkampf misst. Durch die fehlende Anzahl an Metern konnte nicht bestimmt werden, wie viele Zwischenzeiten zu erwarten sind. Zusätzlich ist es offensichtlich, dass beim Stoppen der Zeit regelmäßig versehentlich zu oft oder eventuell zu wenig gedrückt wird. Dadurch entstehen Schwierigkeiten, die der Algorithmus nur schwer lösen kann. Auf jeden Fall müsste hier ein gewisser Aufwand betrieben werden, um den Algorithmus zu trainieren, immer die richtigen Entscheidungen zu treffen. Am einfachsten wäre es an dieser Stelle, die Möglichkeit zu geben manuell Zwischenzeiten hinzuzufügen. Darüber hinaus sind noch die Staffelläufe zu erwähnen, die alle 50m eine Zwischenzeit produzieren und damit eine Ausnahme der Regel bilden.

6.8.4.4 Relative Positionierung

Als der Zielrichteralgorithmus aus unserem Ermessen vollständig und funktionstüchtig implementiert wurde, gab es trotz allem immer wieder neue auftretende Fälle (auch Corner Cases genannt), die nicht beachtet wurden. Allerdings war in diesem bestimmten Fall nicht die schiere Menge an Fällen das Problem, sondern viel mehr die mangelnde Kommunikation zwischen Auftraggeber und -nehmer. Denn die Platzierung wurde zuvor als absolut betrachtet. Dies stellte sich jedoch als Trugschluss heraus und lässt sich leicht an der Tabelle 1 dargestellten Abstimmung erläutern. Hier sind standardmäßig 8 Plätze aufgeführt und es gibt drei Zielrichter. Nach der aktuellen Implementierung des Zielrichteralgorithmus wäre Bahn 3 auf Platz 1 anzusehen. Dies ergibt sich ganz einfach daraus das 2 Zielrichter die Bahn 3 dem Platz 1 zugeordnet haben und damit der Mehrheitsentscheid greift. Relativ gesehen ist dies andererseits nicht korrekt. Wenn man die Reihenfolge aller Bewertungen von Zielrichter 1 betrachtet, ist zu erkennen, dass die Person Bahn 2 vor 3 vor 4 gesehen hat. Bedeutet: Der Zielrichter hat in seiner relativen Reihenfolge die Bahn 2 vor Bahn 3 gesehen und damit ist die Bahn 2 auf Platz 1 anzusehen. Dadurch, dass die Anderen die Bahn 2 nicht in ihrer Bewertung angegeben haben, sie aber Bahn 3 als erstes gesehen und Zielrichter 1 die Bahn 2 vor 3 sieht, ist die Reihenfolge klar bestimmt.

Der Irrtum beruht darauf, dass auch in der Zielrichter-App die Zielrichter ihre Tipps pro Platz abgeben und die Daten auch so (platzbasiert) zum Auswerter übertragen werden. Der 1. Platz heißt in diesem Zusammenhang allerdings nur, dass der Zielrichter die Bahn X für sich persönlich als erstes einlaufen gesehen hat und nicht, dass es zwingend auch der 1. gewesen sein muss. Fazit: Während der „Feuerprobe" der Software gab es jedenfalls kaum Schwierigkeiten bei der Einschätzung der Zielrichter und dem daraus resultierendem berechneten Ergebnis trotz des absolut adressierten Algorithmus. Für selten auftretende Einzelfälle ist der Algorithmus jedenfalls noch nicht ausgereift.

Platzierung	Zielrichter 1	Zielrichter 2	Zielrichter 3
1	2	3	3
2	3	4	4
3	4		
4			
5			
6			
7			
8			

Tabelle 1 Beispielhafte Abstimmung der Zielrichter

6.8.5 Umgang mit der Software

Allgemein gilt, dass die Spalten ZR.Amtl und Zeit Amtl. als einzige bearbeitbar sind. Manchmal kann es nützlich sein bei einem n.E. in der amtlichen Spalte eine Position einzutragen. Aufgrund dessen das bei einem n.E. die Zeiten und einige andere Informationen vorerst verborgen werden. Dies kann im Nachhinein durch die Eingabe einer "0" in die Zelle wieder auf n.E. zurückgesetzt werden. Es können aber auch durchaus Fälle auftreten, in denen es sinnvoll ist, gewisse Bahnen vorerst aus der Wertung zu nehmen. Dies geht wie vorher erwähnt mit der Eingabe einer „0". Alle diese Änderungen werden natürlich in der Änderungshistorie aufgenommen und können dort nachträglich eingesehen werden. Bei einem Editiervorgang ist es wichtig zu beachten, dass eine Eingabe immer mit der Eingabetaste bestätigt werden muss. Da ansonsten das nachfolgende Klicken in eine andere Zelle, die zuvor getätigte Eintragung wieder aufhebt und der alte Wert erneut in der Zelle dargestellt wird.

Sollte es zu unterschieden in der Anzahl der Zwischenzeiten kommen, etwa dass eine Bahn 3 Zeiten hat, einer andere jedoch nur 2, wird ein kleines Kästchen unter der Tabelle rot umrandet. In diesem Kästchen lässt sich ablesen wie viele Zeiten jede Bahn hat.

Die Eingabe der Zeit stellt eine Besonderheit dar, weil hier mehr als nur ein Zeichen eingetragen werden kann und erwartet wird. Das erwartete Format lautet „hh:mm:ss,ms“. Um die Eingabe zu vereinfachen und zu beschleunigen, ist es erlaubt die Doppelpunkte und das Komma wegzulassen, denn diese werden von der entsprechenden Methode automatisch gesetzt. Außerdem wird die Eingabe nach links mit Nullen aufgefüllt. Die Eingabe „12345“ würde wie folgt umgesetzt werden: „00:01:23,45“.

7 Testkonzept

Um das Projekt zu einem erfolgreichen Abschluss zu bringen und immer zu gewährleisten, dass die einzelnen Komponenten sowohl einzeln als auch im Zusammenspiel mit den anderen Anwendungen funktionieren, wurde nach der Methode „test-driven development“ (TDD) vorgegangen. TDD beinhaltet, dass nicht, wie üblich, mit der Implementierung einzelner Funktionen begonnen wird, sondern zuerst Tests geschrieben werden.

Nachdem die einzelnen Schnittstellen definiert sind und bekannt ist, welche Funktionalitäten implementiert werden sollen, werden zu Beginn der Entwicklungsphase JUnit-Tests etc. geschrieben. Diese Tests sind solange fehlerhaft, bis die eigentliche Implementierung korrekt umgesetzt wurde. Dies hat zum Vorteil, dass es keinen ungetesteten Quellcode im Programm gibt und sich der Entwickler auf das Wesentliche konzentriert. Im Idealfall werden die Tests von einem anderen Entwickler geschrieben wie die Implementierung. Dadurch ist sichergestellt, dass eine Funktionalität unterschiedlich betrachtet wird und alle potenziellen Fehlerfälle betrachtet werden.

Neben TDD wird zusätzlich Continuous Integration (CI) verwendet. GitLab unterstützt bereits CI und somit kann dies direkt genutzt werden. Bei Continuous Integration werden nach jeder Änderung, die auf den GitLab-Server commited wird, die zuvor angelegten Tests durchgeführt und eine Rückmeldung gegeben.

Um zu verhindern, dass Funktionalitäten nur in einer bestimmten Umgebung funktionieren (z.B. nur auf dem Rechner des Entwicklers), läuft auf dem GitLab-Server ein sogenannter Docker. Dies ist eine Software zur Isolierung von Anwendungen mit Containervirtualisierungen. Somit wird jedes Einzelprogramm in einem eigenen System getestet, welches nach jeder Änderung komplett neu aufgesetzt wird. Sollte eine Änderung nur unter bestimmten Systemvoraussetzungen funktionieren, wird dies ebenfalls vom System erkannt.

Neben den JUnit-Tests war es ebenfalls nötig bestimmte Funktionalitäten mithilfe von Mock-Ups zu testen. Ein Mock-Up ist eine Abbildung eines Designs oder eines Gerätes. Da die einzelnen Anwendungen isoliert voneinander auf dem GitLab getestet werden, es aber Funktionalitäten gibt, die nur im Zusammenspiel mit einer anderen Anwendung funktionieren, müssen für diese Mock-Ups erstellt werden. Diese Mock-Ups simulieren dann diese Eigenschaften. Ein konkreter Mock-Up in den einzelnen Gruppen war der Mock-Up für die MQTT-Kommunikation. Die einzelnen Anwendungen kommunizieren in der Testumgebung nicht wirklich miteinander, sondern dies muss simuliert werden. Wenn zum Beispiel die Zielrichter-App Informationen über den nächsten Lauf benötigt, wird dies während des Tests vom Mock-Up simuliert und es findet keine echte Kommunikation mit der Anwendung des Kontrollrechners statt.

Während des Projektverlaufs stellt sich heraus, dass aufgrund der geringen Erfahrungen bzgl. Schreiben von Tests und dem somit hohen Entwicklungsaufwand die Implementierung der Tests sehr aufwendig ist. Somit wurde der Fokus von allen Gruppen vermehrt auf die Implementierung gelegt, um den Termin des Schwimmwettkampfes einhalten zu können. Außerdem wurde das Projektmanagement angepasst, sodass hauptsächlich die Prototypen und das Zusammenspiel der einzelnen Komponenten getestet wurde. Dabei sind Fehler aufgefallen, die über das CI auf dem GitLab nicht aufgefallen wären, da es z.B. Probleme mit der Kommunikation über MQTT gab oder die Konfiguration der MQTT-Topics noch nicht komplett korrekt war.

7.1 Zeitmesssystem

Die Funktionen des Zeitmessprogramms wurden mit Hilfe des Unit-Testing-Framework Cmocka getestet, welches nur die Standard C Bibliothek benötigt. Da der Programmcode der Zeitmessung nicht

allzu umfassend ist, gibt es Testfälle für die Überprüfung des Zeitstempels, Anlegen eines Backups, sowie der MQTT-Verbindung.

Die Tests befinden sich in einer eigenen C-Datei und werden mit dem Makefile kompiliert, aber müssen zum Testen exklusiv ausgeführt werden. Hierfür gibt es das Label *test* im Makefile, welches mit *make test* aufgerufen werden kann.

Das Testen der MQTT-Funktionen gestaltete sich als besonders schwierig, da ein Mockup MQTT-Broker für C benötigt wurde und dieser aufwendig zu entwickeln ist. Dies konnte aber durch die Verwendung der Paho MQTT Library umgangen werden, welche eigene Unittests inklusive Mockup beinhaltet und so als Grundlage unserer Tests dienten.

7.2 Zielrichter-App

Um die Zielrichter-App zu testen, wurden zu Beginn auf dem GitLab die entsprechenden Tests eingerichtet. Dazu gehört zum einen der JUnit Test, zum anderen mussten sogenannte „functional tests“ gemacht werden. Eine Teilgruppe beschäftigte sich zu Beginn des Projektes damit, die entsprechenden Tests auf dem GitLab einzurichten. Dazu musste die gitlab.yml-Datei entsprechend angepasst werden.

Der Idealfall hätte nach korrekter Modifikation der Datei dann folgendermaßen ausgesehen: Nach einem Commit eines Nutzers wären alle notwendigen Komponenten neu geladen worden, sodass das gesamte System unbeeinflusst genutzt wird. Danach wären die geschriebenen JUnit-Tests getestet worden und anschließend wäre auf dem Server ein ebenfalls neu mitinstallierter Android-Emulator verwendet worden, um die „functional tests“ durchzuführen. Da kein Gruppenmitglied Erfahrungen mit diesem System hat und nach diversen Anpassungen der yml-Datei und entsprechender Recherche, dieser Ablauf nicht einwandfrei lief, wurde dies in die Zuständigkeit von Herrn Gosewehr übertragen. Somit konnte sich unsere Gruppe auf die eigentlichen Tests und die Implementierung fokussieren.

Schon zu Beginn des Projektes, stellte sich heraus, dass für die Android-Anwendung kaum JUnit-Tests notwendig sein werden. Da bei einer Android-App die meiste Funktionalität an den Activities implementiert wird, werden diese direkt bei den UI-Tests mitgetestet. Die Activities sind in diesem Fall die eigentlichen Fenster, die der Nutzer sieht und mit denen er interagieren kann.

Die UI-Tests sind Tests, die den Bereich der „functional tests“ abdecken sollen. UI steht für User Interface und somit wird über diese Tests bereits die Funktionalität der einzelnen Aktivitäten geprüft. Nach kurzer Recherche wurde festgelegt, dass zur Umsetzung dieser Tests, das in Android bereits integrierte, Espresso verwendet wird. Mit dieser Software ist es möglich, sobald eine entsprechende grafische Oberfläche vorhanden ist, einen Test aufzuzeichnen. Dieser kann dann später auf dem Emulator, der auf dem GitLab-Server läuft, ausgeführt werden.

Für die Aufzeichnung muss das Android-Gerät am PC angeschlossen werden. Nach dem Starten der Aufzeichnung, wird die zu testende Software, in diesem Fall die Zielrichter-App, gestartet und es können Aktionen durchgeführt werden. Zum Beispiel kann der Entwickler im Login-Fenster den Login-Code eingeben und wird auf die Hauptseite weitergeleitet. In einem Pop-Up-Fenster auf dem Computer kann der Entwickler diese Aktionen sehen und auch entsprechende Prüfungen einfügen. So kann zum Beispiel geprüft werden, ob nach dem Login die Hauptseite sichtbar ist oder ob bestimmte Fehlertexte existieren, wenn der Login fehlgeschlagen ist.

Sobald der Testdurchlauf abgeschlossen wurde, kann dieser gespeichert werden. Android Espresso generiert dann den Quellcode, der die Nutzerinteraktion und die Prüfungen wiederspiegelt und legt diesen im Testverzeichnis ab. Anschließend kann der Tests wie ein JUnit-Test ausgeführt werden. Dazu muss weiterhin ein Gerät angeschlossen sein oder ein Emulator gestartet sein. Dort wird die

Anwendung dann gestartet und die zuvor „aufgenommenen“ Aktionen werden von selbst durchgeführt. Sind die implementierten Prüfungen positiv, ist der Test erfolgreich. Schlägt der Test fehl, da z.B. eine Funktionalität noch nicht implementiert wurde, dann weiß der Entwickler, dass hier noch Entwicklungsbedarf besteht.

Um die einzelnen Tests immer möglichst klein zu halten, wurden die zuvor ermittelten Testfälle nacheinander aufgenommen. Anhand bestimmter Faktoren wurde geprüft, ob die Anwendung korrekt reagiert hat oder nicht. Bei diesen Tests war es allerdings immer notwendig die komplette Anwendung zu durchlaufen. Es konnte z.B. nicht nur das Abgabefenster getestet werden. Die Anwendung startet immer beim Login-Fenster und die Fenster müssen soweit bearbeitet werden, bis das Abgabefenster geprüft werden kann. Für die Aufnahme des Tests muss die grafische Oberfläche bereits implementiert worden sein. Somit war diese Teilgruppe immer abhängig von der Teilgruppe, die parallel die Implementierung vorgenommen hat.

Der Aspekt des Test-Driven-Developments, dass die Tests vorab unabhängig von der Implementierung entwickelt werden, konnte somit aufgrund der Randbedingungen nicht komplett eingehalten werden. Es wurde zwar immer darauf geachtet, dass die Reihenfolge „grafische Oberflächen implementieren → UI-Tests aufnehmen → Funktionalität umsetzen“ so gut wie möglich eingehalten wurde. Dies war allerdings nicht immer möglich, da bestimmte Funktionalitäten erst nach der Implementierung korrekt aufgezeichnet werden konnten.

Neben der Abhängigkeitsproblematik gab es zudem Probleme mit Espresso selbst, da die aufgezeichneten Tests nicht immer kompilierfähig waren. Dies umfasste kleinere Fehler, wie zum Beispiel das Importieren einer falschen Bibliothek und größere Fehler, wie zum Beispiel Probleme mit der GUI-Hierarchie, sodass manche Tests nicht korrekt geprüft wurden. Dadurch waren hier weitere Korrekturarbeiten von Seiten des Entwicklers notwendig. Außerdem ist die Verwendung eines Android-Emulators auf dem GitLab-Server sehr zeitaufwendig, mit viel Recherche verbunden und die Fehlersuche ist schwierig, sodass am Ende des Projektes die UI-Tests nicht auf dem Server durchliefen.

Zur Mitte des Projektes wurde die eigentliche Entwicklung vom TDD hin zu einer Prototypentwicklung umgestellt. Aus diesem Grund hat man sich ab dann zunächst gezielt auf die Implementierung der Funktionalitäten und dem manuellen Test der Anwendung mit den anderen Gruppen konzentriert. Somit sind viele Anpassungen gemacht worden, die innerhalb der Gruppe mit den anderen Komponenten erfolgreich getestet worden sind, die aber dafür gesorgt haben, dass viele der zuvor geschriebenen Tests nicht mehr erfolgreich durchliefen.

Das Gesamtsystem und die Funktionalitäten sind von allen Gruppen im Zusammenspiel getestet worden. Mögliche Fehler, die den Gesamtablauf beeinflussen könnten, wurden über diese Tests herausgefiltert und konnten korrigiert werden. Dass die zuvor erstellten Tests nach der größeren Anpassungsphase kurz vor dem Schwimmwettkampf nicht mehr erfolgreich sind, wurde nicht erkannt, da bis zum Ende des Projektes die UI-Tests auf dem GitLab-Server nicht ausgeführt werden konnten.

Wären die UI-Tests erfolgreich eingerichtet worden, hätte es bereits beim push auf den Server eine Fehlermeldung gegeben. In diesem Fall wäre eine Fehlersuche einfacher gewesen, da sich das Finden der Fehlerursache im Nachhinein als zeitintensiv und aufwendig herausgestellt hat. Es wäre ebenfalls möglich gewesen die Tests manuell durchzuführen und dann festzustellen, dass eine Änderung fehlerhaft ist. Dies ist allerdings ebenfalls zeitaufwendig, weil die Tests einzeln aufgerufen und dann mithilfe eines angeschlossenen Gerätes getestet werden müssten. In der Phase kurz vor dem Schwimmwettkampf wurde dies ebenfalls nicht gemacht, da der mögliche Fehler keinen Einfluss auf den eigentlichen Ablauf hat und bei den Tests mit den anderen Komponenten weder aufgefallen noch für ein Problem gesorgt hat.

7.3 Kontrollprogramm

Damit das Kontrollprogramm getestet werden kann, wurden zuerst Testfälle überlegt und im GitLab dokumentiert. Es wurden Test erstellt für die Kommunikation mit MQTT und JUnit Test für die JavaFX-Elemente (kurz. TestFX). Für das Testen von MQTT wurde ein eigenes Testboard erstellt.

Das Vorgehen des Test Driven Developments, dass zuerst der Test erstellt und dann die Funktion implementiert wird, konnte während der Entwicklung nicht konsequent durchgeführt werden. Mit nur zwei Mitgliedern in der Gruppe und den immer wachsenden Anforderungen an das Programm, blieb meist nicht die Zeit, sich zuvor noch Gedanken über Testfälle und die Erstellung des Tests zu machen.

7.3.1 MQTT Testboard

Um eine Verbindung vom Kontrollprogramm und dem Testboard zu gewährleisten, muss zuerst im Quelltext vom Testboard die IP-Adresse des Kontrollrechners angegeben werden. Das Testboard ist in drei Segmente aufgeteilt: Zeitnehmer, Zielrichter und Auswerter.

Im linken Segment bilden die Buttons “Start” und “Drücker 1.1” bis “Dücker 8.2” die Startanlage sowie die einzelnen Zeitnehmer ab. Nach Klick auf den Buttons wird ein Zeitstempel auf die entsprechenden Topics gesendet.

Im mittleren Segment werden die Zielrichter dargestellt. Im Bereich “Allgemein” kann ein neuer Lauf angefordert werden, indem die Wettkampf-Nr. und die Lauf-Nr. eingegeben werden. Außerdem kann ein Master-Zielrichter simuliert und das Beenden des Laufes erzwungen werden. In den folgenden Bereichen werden drei Zielrichter simuliert. Die Anzahl ist beliebig erweiterbar. Die Token müssen dafür dem Kontrollprogramm entnommen werden. Dann kann man den jeweiligen Zielrichter an- und abmelden sowie Ergebnisse senden. Die Ergebnisse enthalten lediglich Dummy-Daten, bei denen nur die Wettkampf-Daten korrekt sind, da der Kontrollrechner diese ansonsten nicht akzeptieren würde. Alle gesendeten und empfangenen Zeichenketten der Zielrichter können in den unteren Textfeldern eingesehen werden.

Sowohl von den Zeitdrückern als auch von den Zielrichtern, die angemeldet sind, werden alle 5 Sekunden Status-Updates gemacht.

Im rechten Segment kann überprüft werden, welche Nachricht an den Auswerter gesendet wurde.

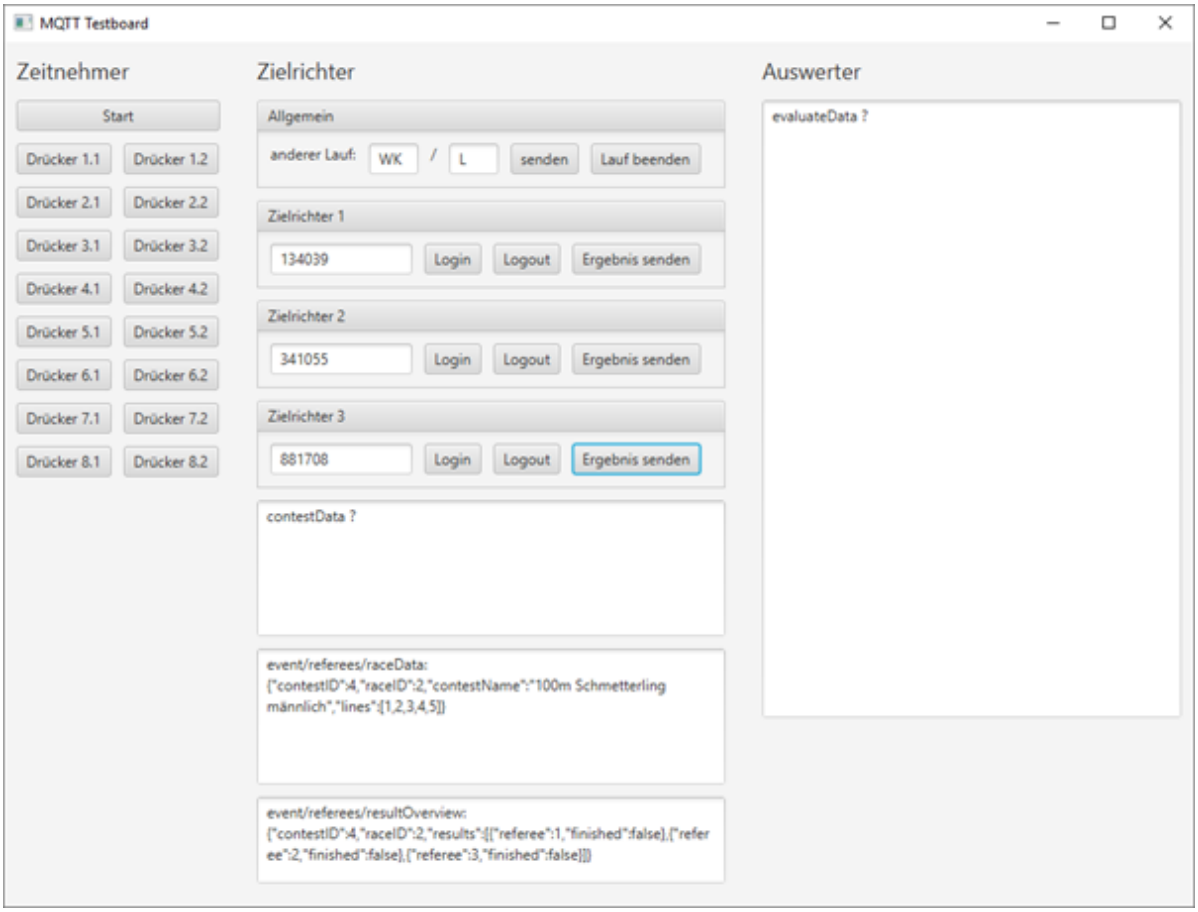


Abbildung 47 MQTT Testboard

7.3.2 TestFX

Mit TestFX können Oberflächen Tests geschrieben werden. Die TestFX Tests wurden meistens eher zweitrangig behandelt, da sich die Benutzeroberfläche oft im Laufe des Projektes geändert hatte. Viele Tests konnten auch nicht geschrieben werden, da das meiste im Kontrollprogramm mit MQTT in Verbindung steht. Die geschrieben Tests bilden einen Gesamtablauf ohne einer MQTT Verbindung ab. Dieser Gesamtablauf bildet u.a. einen Test von der Generierung eines neuen Tokens, das Verbinden des Zielrichters mit Abfrage der Status Farbe und das Beenden eines Laufs in der detaillierten Wettkampfübersicht. Dabei klickt TestFX automatisch auf die angegebenen JavaFX Elemente. Es werden Fehler geworfen, falls diese Elemente nicht existieren und es zu einem nicht erwarteten Ergebnis kommt.

7.4 Auswertungsprogramm

Da das Auswerterprogramm in Java geschrieben wurde, ließ sich Test-Driven-Development mithilfe vieler verschiedener Frameworks realisieren. Es wurde sich für JUnit und TestFX entschieden. JUnit bietet Möglichkeiten, um grundlegende Java-Methoden zu testen. TestFX erlaubt es auch JavaFX Inhalte zu testen. Neben dem ständigen lokalen Testen sollte das Produkt auch nach jedem Git-Commit auf einem Server getestet werden. Dieser Server enthielt ein Docker-Image, welches eine saubere Umgebung ohne Fremdeinwirkung von anderen Programmen sicherstellt. So lässt sich zuverlässig bestimmen, welche Komponenten für einen einwandfreien Betrieb der Software notwendig sind. Ein Fall, bei dem eine Lösung nur funktioniert, weil zufällig eine fremde Software lokal auf dem Computer des Programmierers installiert ist, fällt hier sofort auf.

Die gesamte Gruppe hat versucht, sich an die Anforderungen von TDD zu halten. Jedoch wurde auch schnell deutlich, dass dies keine leichte Aufgabe ist. Nicht nur mussten sich alle Gruppenmitglieder in JavaFX und zu einem gewissen Grad auch in komplexere Java-Themen einarbeiten, gleichzeitig sollten Tests immer vorher geschrieben werden. Oft war JavaFX noch nicht in einem Maße verstanden, sodass Tests dafür im Voraus geschrieben werden konnten. Viele Klassen und Methoden stellten sich im Verlauf des Projekts als falsch oder unvollständig heraus und mussten aufwändig geändert werden, was auch zur Folge hatte, dass alle dafür geschriebenen Tests nicht mehr funktionierten. Gewöhnliche Java-Algorithmen ließen sich jedoch gut mit JUnit testen und dies war auch der meist gewählte Weg. Für den Zielrichter-Algorithmus zur Bestimmung des Mehrheitsentscheids wurden zwar nicht zuerst Tests geschrieben, jedoch sind auch durch nachträglich geschriebene Tests einige Corner-Cases aufgefallen.

Ein weiteres Problem stellte aber auch das Docker-Image auf dem Gitlab-Server dar. Die Anwendung wurde mit Oracle Java 8 geschrieben. JavaFX ist in Oracle Java 8 noch Teil der Standardbibliothek, jedoch nicht in Open-Source-Implementierungen von Java wie bspw. OpenJDK. Oracle Java und OpenJDK sind in dieser Hinsicht nicht kompatibel. Da es jedoch keine Möglichkeit gab Oracle Java in das Docker-Image zu integrieren, wurde hier mit OpenJDK und dem Zusatzpaket OpenJFX gearbeitet. Es ist schon jetzt ersichtlich, dass es sich um verschiedene Laufzeitumgebungen handelt. Anfangs, als noch keine komplexen Sachverhalte mit JavaFX programmiert wurden, funktionierten alle Tests auf dem Docker-Image. Im Laufe des Projekts änderte sich dieser Sachverhalt allerdings. Auf dem Docker-Image schlugen Tests aus unbekannten Gründen fehl, obwohl sie auf verschiedenen unabhängigen Maschinen, mit teilweise unterschiedlichen Betriebssystemen, funktionierten. Da auf die Tests im Docker-Image somit kaum noch Verlass war und die Deadline immer näher rückte, gerieten die Tests immer weiter in den Hintergrund.

7.5. Lenex und Datenbank

Sowohl die Lenex (Rück-)Importierfunktionen als auch die Datenbanken eignen sich tatsächlich ziemlich gut für Test-Driven-Development. Beide Systemkomponenten sind nicht eigenständig am Laufen und es eignet sich schon sehr gut von Beginn an, einzelne Komponenten zu testen. Sowohl die Datenbank als auch der Lenex-Parser wurden in Java geschrieben auf Basis des Spring-Frameworks, welches beim Testen aufgrund der @Autowired-Annotation einiges erleichtern kann. Die Tests wurden größtenteils in der White-Box Ansicht gefertigt, schließlich ist das Wissen über die internen Funktionen bekannt. Mittels der Tests kann festgestellt werden, ob Transaktionen erfolgreich abgehandelt werden können, ob die Relationen zwischen den Tabellen der Datenbanken funktionieren, ob alle oder gewisse Daten einer Lenex-Datei geladen werden können oder andere semantische Checks. Bei nicht eigenständigen Systemkomponenten eignet sich Test-Driven-Development am besten, um auch während der Entwicklung Fehler festzustellen und diese auszumerzen.

8 Projektverlauf

Das Projekt startete, wie bereits erwähnt, mit einer zweiwöchigen Intensivphase. Dort wurde zunächst durch jeden Teilnehmer ein Thema der vorherigen Projektgruppe vorgestellt, um den Einstieg in das Projekt zu erleichtern und bereits erste Diskussionen für die weitere Vorgehensweise auszulösen. In dieser Phase wurde dann auch der vorläufige Projektleiter gewählt und besprochen, nach welcher Methode das Projektmanagement (siehe Abschnitt 3 Projektmanagement) ausgelegt wird.

Der Fertigstellungstermin des Projekts wurde vorerst nicht genau festgelegt, aber das System sollte auf jeden Fall vor dem 07.12.18 fertiggestellt sein, da es am 07.12.18 und 08.12.18 beim Adventsschwimmen in der Friesentherme Emden eingesetzt werden sollte.

Im weiteren Verlauf der Intensivphase wurde das Systemkonzept (siehe Abschnitt 4 Systemkonzept) erstellt und die ersten Anforderungen im Pflichtenheft festgehalten. Danach wurde die Gruppe in einzelne Teilgruppen unterteilt, die sich jeweils mit einer Teilkomponente des Systems beschäftigen sollen. Nach der Einteilung der einzelnen Gruppen, haben diese sich jeweils in die Anforderungen der betreffenden Komponente eingearbeitet und daraufhin recherchiert, welche Entwicklungsumgebung, Frameworks und sonstige Anforderungen benötigt werden. Die Ergebnisse der Recherchen wurden immer wieder mit der gesamten Gruppe zu Beginn des Termins besprochen.

Auch sollte jede Teilgruppe eigene Meilensteine definieren, um nicht im Nachhinein das Ziel aus den Augen zu verlieren. Die Meilensteine und auch sonstige wichtige Informationen, wie z.B. die Topics der MQTT Schnittstellen und dazugehörigen JSON-Formate, wurden im Versionsverwaltungstool Gitlab festgehalten. Somit hatte jede Gruppe Zugriff auf alle relevanten Informationen und auch der aktuellste Quellcode war immer gesichert. Auch war es dadurch möglich, dass mehrere Teilnehmer einer Gruppe gleichzeitig an dem Quellcode arbeiten konnten.

Nach der Intensivphase organisierten sich die einzelnen Teilgruppen untereinander selbst und einmal wöchentlich gab es dann eine Besprechung über die Fortschritte der Woche mit den Betreuern und allen Teilnehmern. Zusätzlich wurde noch eine WhatsApp Gruppe eingerichtet, um wichtige Informationen schnell untereinander austauschen zu können.

Es stellte sich aber nach einiger Zeit heraus, dass die Meilensteine nicht genau gehalten werden konnten und die ganze Vorgehensweise nicht für das Projekt passte.

Gegen Mitte Oktober wurde die Rolle des Projektleiters auf eine andere Person übergeben. Zusätzlich wurde nun ein erster Prototyp fertiggestellt, der schrittweise Woche für Woche erweitert wurde. Dadurch konnten immer wieder Fehler festgestellt und rechtzeitig behoben werden. Auch gab es innerhalb der gesamten Gruppe neue Motivation, weiter zu entwickeln, da jetzt das gesamte System immer wieder getestet wurde und ein Fortschritt erkannt wurde.

Auch gab es zwischendurch immer mal wieder situationsbedingte Skype Meetings (zusätzlich zu den wöchentlichen Projektmeetings in Präsenzform) mit immer mindestens einem Teilnehmer pro Teilgruppe, um weitere Vorgehensweisen und mögliche Schwierigkeiten zu klären. Später hat sich die Gruppe auch noch sehr oft zusätzlich ohne die Betreuer getroffen, um weitere Tests mit dem Gesamtsystem durchzuführen. Durch diese Maßnahmen konnte immer wieder rechtzeitig sichergestellt werden, dass die einzelnen Komponenten nicht unabhängig voneinander entwickelt werden und die Kommunikation der Teilnehmer untereinander sich nicht nur auf die einzelnen Teilgruppen beschränkt.

Ende November wurde das System dem ersten Vorsitzenden des Schwimmvereins Neptun Emden vorgestellt. Dabei wurde die Gruppe noch auf weitere Punkte hingewiesen, die noch angepasst werden mussten. Nach der Anpassung besuchte er uns noch einmal und testete das gesamte System einmal

mit der ganzen Gruppe durch, auch um realistisch einschätzen zu können, ob das System wirklich bereit dazu ist, auf dem Schwimmwettkampf eingesetzt werden zu können.

Das System wurde dann zum 03.12.2018 fertiggestellt und danach folgten nur noch weitere Tests des Gesamtsystems, um Fehler zu beheben und es möglichst stabil laufen zu lassen. Dabei wurde auch teilweise bis zu vier Stunden am Stück ein Test durchgeführt mit allen möglichen Szenarien, die beim Schwimmwettkampf auftreten können.

Am 07.12. und 08.12. konnte das System dann erfolgreich bei dem Schwimmwettkampf eingesetzt werden. Es traten gelegentlich Fehler auf, die aber entweder schnell behoben oder durch genaues Aufpassen der Gruppenmitglieder verhindert werden konnten. Alle diese Fehler wurden dann in einer Nachbearbeitung angepasst, sodass jetzt ein stabiles System entstanden ist, welches aber natürlich noch erweitert werden kann. Das gesamte System wurde dann zu Ende des Projekts noch einmal komplett getestet, um auszuschließen, dass letzte Änderungen noch fatale Folgen für das System hatten. Zusätzlich wurde dann von allen Teilnehmern noch eine Code Review durchgeführt, um sicherzustellen, dass alle Schnittstellen sauber zusammenpassen und eine saubere Programmierung mit ausreichender Kommentierung vorliegt.

8.1 Testeinsatz beim Adventsschwimmen

Am 7. und 8. Dezember 2018 veranstaltete der Schwimmverein Neptun Emden das 51. internationale Adventsschwimmfest in der Friesentherme Emden. Dies war der erste Testlauf des Systems unter realen Bedingungen. Am ersten Tag der Veranstaltung wurden Langstrecken geschwommen, wodurch ein ruhiger Ablauf der Veranstaltung entstand und es nur zu Bedienfehlern seitens der Zielrichter und den Zeitnehmern kam. Der zweite Wettkampftag wurde durch die Wettkämpfe über die kürzeren Distanzen und die Überkopfstarts hektischer. Vor allem die Überkopfstarts, bei denen sich die Schwimmer noch im Wasser befinden, während das nächste Rennen gestartet wird, sorgten für ein Problem zwischen Zielrichter-App und Kontrollrechner.

Insgesamt war das Schwimmfest ein erfolgreicher Testlauf für die Projektgruppe und den Organisatoren des SV Neptun Emden, indem wichtige Erkenntnisse für den weiteren Projektverlauf gewonnen werden konnten.

8.1.1 Zeitmesssystem

Da zum Zeitpunkt des Adventsschwimmen noch die alten Gehäuse der Raspberry Pis genutzt wurden und keinen entsprechenden Schutz vor Wasser boten, wurden diese für den Wettkampf in Plastiktüten gehüllt und an den Startblöcken mit Kabelbindern befestigt.

Durch die einfache Bedienung der Drücker konnten über den gesamten Wettkampfverlauf fast alle Endzeiten gestoppt werden. Ausfälle von Zeiten kamen nur zustande, wenn die beteiligten Zeitnehmer auf Grund der Gewohnheit der Stoppuhr vergaßen, die Drücker zu betätigen. Infolge zweier fehlender Zwischenzeiten eines Raspberry Pis wurde festgestellt, dass ein Akkuwechsel vorgenommen werden musste. Dies führt wegen der Langstrecken aber zu keiner verpassten Endzeit.

Aus dem Testlauf kann resümiert werden, dass die Raspberry Pis zukünftig ihren Status und/oder Akkustand dem Kontrollrechner mitteilen sollten, damit bei Ausfällen rechtzeitig auf diese reagiert werden können.

8.1.2 Zielrichter-App

Am Freitag den 07.12.2018, waren beim Adventsschwimmen drei Zielrichter vor Ort. Diese Zielrichter bekamen kurz vor dem Start des ersten Wettkampfes eine kleine Einführung in die App. Hier wurde ihnen erklärt wie sie die App bedienen müssen. Da es schon allgemein die Runde gemacht hatte, dass die Zielrichter ein „Handyspiel“ bekommen würden, war die Stimmung positiv und damit auch das

Interesse an der App sehr groß. Dadurch verliefen die Erklärungen zur Bedienung sehr entspannt und wurden auch gut verstanden. Trotzdem waren einige skeptisch, ob die Bedienung wirklich so reibungslos ablaufen würde. Vor allem die Zielrichter, die nicht so oft ein Smartphone bedienen, hatten große Bedenken. Da die Zielrichtergruppe aber versicherte, dass sie bei den ersten Durchgängen Hilfe leisten würden, legte sich diese Aufregung auch wieder recht schnell.

Als dann der erste Lauf beendet war, sollten die Zielrichter ihre Abgaben tätigen. Dies verlief fast ohne große Erklärungen, da das Bedienen des Drag und Drop sehr intuitiv ist und damit alle gut zurechtkamen.

Im Verlaufe des Abends entstanden dennoch einige Probleme. Ab und zu loggte sich ein Zielrichter unabsichtlich aus. Dies musste von der Zielrichtergruppe behoben werden, da die Zielrichter nicht wussten was zu tun sei. Auch war hier auffällig, dass die Smartphones mit viel Bedacht und Vorsicht benutzt wurden. Das führte dazu, dass selbst wenn der Bildschirm nur ausging bzw. gesperrt wurde, die Zielrichter die Gruppenmitglieder gerufen haben, da sie von selbst nicht wussten wie sie den Bildschirm wieder entsperren.

Ein weiteres Problem bestand darin, dass durch das Spritzwasser der Schwimmer, einige Tropfen die Smartphones trafen. Dadurch konnte das Drag und Drop nur noch sehr schwer von den Zielrichtern benutzt werden, nicht zuletzt auch, weil sie sich nicht trauten die Bildschirme abzuwischen. Dies mussten die Mitglieder der Zielrichter-Gruppe erledigen.

Die Zielrichter gaben auch Feedback zur App. So fanden sie, dass der Weiter-Screen unnötig ist bzw. sie erwarteten, dass dieser von selbst verschwindet und nicht durch ein erneutes Bestätigen. Der Weiter-Screen wurde daraufhin, nach dem Adventsschwimmen, entfernt.

Am Samstag den 08.12.2018, waren beim zweiten Tag des Adventsschwimmens, fünf Zielrichter vor Ort. Es waren nicht die gleichen wie am Tag zuvor. Der Anfang verlief ähnlich zu dem am Vortag. Die Zielrichter wurden eingewiesen und bekamen die Smartphones.

Auch hier verlief bei den ersten Durchgängen alles ohne große Probleme, da die Gruppe Hilfestellung bot und damit den Einstieg erleichterte.

Im Verlauf des Abends traten ähnliche Probleme wie am Tag zuvor auf. Zusätzlich hatten die Zielrichter aber noch Schwierigkeiten damit, die Smartphones zu bedienen und gleichzeitig die Zettel auszufüllen, da beides zur Sicherheit noch erfasst werden sollte. Dies trat am Vortag nicht auf, da dort die Intensität der Wettkämpfe nicht so hoch war. Auch hatten die Zielrichter Probleme damit, das Smartphone, während es nicht im Gebrauch war, irgendwo abzulegen. So entstanden Situationen, wo das Smartphone fast heruntergefallen oder sogar fast im Schwimmbecken gelandet wäre. Hier wäre eine Handyhalterung am Stuhl nützlich gewesen.

Das Feedback der Zielrichter war ebenfalls ähnlich zu den vorherigen. Dazu kam, dass ein Zielrichter es gerne gehabt hätte, dass zusätzlich zu dem Drag und Drop, ein Klicken auf die Bahnen und automatischen zuordnen auf die Positionen möglich gewesen wäre. Diese Idee wurde von der Gruppe angefangen umzusetzen, aber die letztendliche Version beinhaltet noch ein paar Fehler, die noch ausgebessert werden müssen.

Die Resonanz der Zielrichter war aber im Ganzen sehr positiv. Sie sprachen davon, dass es ihnen Spaß gemacht hätte und sie so ein System gerne weiter nutzen würden.

8.1.3 Kontrollprogramm

Das Kontrollprogramm wurde auf einem WIndows-10-Recher direkt aus Eclipse gestartet. Nach Programmstart und Sicherstellung der Verbindung mit allen Geräten, konnten die Wettkämpfe

gestartet werden. Während des Einsatzes sind einige Bugs und Features aufgefallen, die im Folgenden beschrieben werden.

Uns wurde die Wettkampdatei im Lenex-Fomat vom Verein bereitgestellt. Nach Import gab es einige Läufe, die leer waren und die ID 0 hatten. Da diese Läufe also nicht existierten, mussten sie nach hinten verschoben werden, um sicherzustellen, dass sie nicht geladen werden. Nachher wurde eine Prüfung eingebaut, die nur Läufe importiert, wenn die ID nicht 0 ist.

Auch kam es während des Wettkampfes zu dem Fall, dass ein Zeitnehmer die Verbindung verloren hatte und keine Zeit mehr senden konnte. Dieser Fall war der Anlass für die Status-Anzeige, mittels der man auf einen Blick erkennen kann, wenn ein Zielrichter oder Zeitnehmer die Verbindung verloren hat.

Besonders bei den Staffelläufen wäre es wünschenswert gewesen, direkt nach Ende des Laufes die Zeiten der Schwimmer zu sehen, damit die Siegerehrung direkt am Becken (beim Kontrollrechner) durchgeführt werden kann. Diese Funktion wurde vorher nicht gefordert und wurde nicht implementiert. Im Nachhinein wurden die Zeiten in den Details der Wettkampfübersicht ergänzt.

Weiterhin wäre eine weitere Möglichkeit, einen Drucker einzubinden, sodass Zeiten sofort ausgedruckt werde können.

Der größte Bug, der erst im Einsatz bemerkt wurde, war, dass ein Lauf direkt als beendet galt, wenn nach dem Startsignal eines neuen Laufes und in der eingestellten Nachlaufzeit der letzte Zielrichter abgegeben hat. Es lief weiterhin ein Countdown, der den ersten Lauf beenden sollte, aber durch die Abgabe des letzten Zielrichters wurde dieser Lauf jedoch schon als beendet markiert. Somit rutschte der Lauf 2 in die Position 1 und der Countdown entfernte diesen neuen Lauf, wenn es bei 0 angekommen war. Dieser Bug wurde zum einen durch eine Anpassung in der Definition, wann ein Lauf als beendet gilt, und zum anderen mit einer Anpassung im Countdown behoben. Ein Lauf galt zuvor als beendet, wenn die Ergebnisse aller Zielrichter erhalten wurden – nun werden die Zielrichter unabhängig von den Zeitnehmern behandelt und der Lauf ist erst beim nächsten Startsignal (zuzüglich Nachlaufzeit) oder nach manuellem Beenden beendet. Wenn dann schon alle Ergebnisse vorliegen, können die Daten gleich an den Auswerter weitergegeben werden, andernfalls muss auf die Zielrichter gewartet werden. Außerdem wird während des Countdowns auf Änderungen im Beenden Lauf insofern reagiert, dass der Countdown abgebrochen wird, wenn der Lauf sich in dem Index ändert.

Am nächsten Tag konnte das Backup nicht vollständig wiederhergestellt werden, weil Objekte zu früh oder zu spät geladen wurden, sodass u.a. das Log und die Tabelle der aktuellen Läufe wieder leer waren. Nach Anpassung der Objekte und der zeitlichen Reihenfolge der Methodenaufrufe können nun alle Daten, die im Backup hinterlegt sind, geladen und angezeigt werden.

8.1.4 Auswertungsprogramm

Das Auswerterprogramm lief bei dem Adventsschwimmen vom 07. und 08.12.2018 auf dem Laptop von Prof. Dr. Rabe. Lediglich nötig, um das Programm starten zu können, war ein Update der Java-Runtime-Environment auf Java 8 v181. Das Programm wurde außerdem über die Kommandozeile gestartet, um zusätzliche Debug-Meldungen einsehen zu können, falls es zu Problemen kam.

Der Plan sah vor, dass sich der Laptop von Prof. Dr. Rabe in zwei WLAN-Netzwerken gleichzeitig befindet. Genutzt werden sollte dafür ein USB-WLAN-Adapter, sowie der im Laptop integrierte WLAN Chip. Ein Netz dient der Kommunikation mit dem Kontrollrechner, das zweite Netz wird für ein Netzlaufwerk benötigt. Das Netzlaufwerk befindet sich auf einem weiteren, von diesem Projekt unabhängigen, Laptop. Auf diesem Netzlaufwerk sollten alle Exporte aus dem Auswerterprogramm gespeichert werden damit allen Organisatoren bekannt war, wie weit die Auswertung vorangeschritten ist. Nach dem Speichervorgang sollte dann mithilfe des FileChangeListeners eine

ALGE simuliert werden, die EasyWK vorspielt, es sei mit einer echten ALGE-Anlage verbunden. Danach sollte es möglich sein, die Dateien in EasyWK über die ALGE-Schnittstelle zu importieren. Dieser Plan sollte für beide Tage gelten.

Während der Aufbauarbeiten am ersten Tag des Adventsschwimmen, kam es zu Störungen mit dem WLAN-Signal. Der USB-WLAN-Adapter hatte zu wenig Leistung, da sich das Laptop in einer ungünstigen Position befand. Über den WLAN-Adapter sollte ursprünglich die Verbindung zu dem WLAN für das Netzlaufwerk hergestellt werden. Die Verbindung über den internen Chip mit dem WLAN, in dem sich der Kontrollrechner befand, machte keine Probleme. Aus diesem Grund wurde beschlossen ein Netzwerkkabel zu verlegen. Da dies schnell passieren musste, wurde das Kabel durch einen Gang verlegt und mit Klebeband am Boden stolpersicher befestigt. Das ausführbare Auswerterprogramm wurde auf dem Netzlaufwerk abgelegt, da die exportierten Daten immer im selben Ordner abgelegt werden und es keine Funktion gab, um einen anderen Speicherort zu bestimmen.

Als alles aufgebaut war, wurden in Zusammenarbeit mit der restlichen Gruppe wenige kurze Tests durchgeführt, um sicherzugehen, dass eine Verbindung mit allen Komponenten besteht.

Nachdem die ersten Ergebnisse reinkamen, konnte die Auswertung beginnen. Am ersten Tag wurden nur wenige, dafür lange Wettkämpfe geschwommen. Die Auswertung musste also nicht in einem hohen Tempo erfolgen und funktionierte in fast allen Belangen wie geplant. Allerdings gab es auch einige Probleme. Es war nicht bei jedem Wettkampf möglich alle Zwischenzeiten anzuzeigen. Klickte man auf den Button, war ein Fehler in der Konsole zu sehen. Dies führte glücklicherweise jedoch nicht zum Absturz des Programms. Bei der manuellen Eingabe einer amtlichen Zeit mangelte es außerdem an Eingabekomfort. Die Zeit wurde im Format hh:mm:ss,msms erwartet und konnte auch nur genau so eingegeben werden. Wollte man also eine Zeit von einer Minute eingeben, musste „00:01:00,00“ eingeben werden. Prof. Dr. Rabe wünschte sich die Möglichkeit dieselbe Zeit auch als „10000“ eingeben zu können, wie es in EasyWK auch möglich ist. Beide Probleme wurden im Anschluss an den Praxistest behoben.

Am zweiten Tag standen sehr viel mehr kurze Schwimmwettkämpfe an. So konnte getestet werden, ob das Produkt diesem Druck standhalten kann. Glücklicherweise war das Produkt der Herausforderung gewachsen und bis auf die bereits oben genannten Probleme, traten keine weiteren auf. Sobald ein Wettkampf reinkam, der kein eindeutiges Ergebnis hatte, musste händisch analysiert werden, wie sich das echte Ergebnis zusammensetzt. Währenddessen kamen immer wieder neue Wettkämpfe rein welche auf eine Auswertung warteten. Dieser Rückstau konnte jedoch immer wieder aufgeholt werden.

Abschließend lässt sich sagen, dass das Auswerterprogramm seine Aufgabe, bis auf wenige Kinderkrankheiten, gut erfüllt hat. Außerdem war das Adventsschwimmen ein sehr guter Praxistest, aus dem Erkenntnisse für Verbesserungen gewonnen werden konnten. So wurde u.a. der bereits angesprochene Fehler beim Aufruf von Zwischenzeiten behoben, zusätzlich aber auch neue Features eingebaut. So lässt sich die amtliche Zeit in der finalen Version sehr viel anwenderfreundlicher eingeben und zusätzlich wurde die Übersicht um ein kleines Fenster ergänzt, in dem sofort ersichtlich ist, ob alle Bahnen dieselbe Anzahl Zwischenzeiten haben.

9 Ausblick

9.1 Weiterentwicklung und Features

9.1.1 Zeitmesssystem

Im Laufe des Projekts kamen immer wieder neue Ideen und Erweiterungen für das Zeitmesssystem ins Gespräch. Die neuen Gehäuse würden sich durch das Fenster perfekt für ein Display eignen. Auf diesem könnte wie bei einer Stoppuhr die genommene Zeit angezeigt werden und zum Beispiel auch die letzten zwei Zwischenzeiten. Zusätzlich könnten Anzeigen für den Akkustand und die WLAN-Verbindung implementiert werden. Außerdem wäre es möglich, vor dem Start die zu schwimmende Disziplin und Strecke sowie den Namen des Schwimmers zur Vermeidung von Verwechselung auf dem Monitor anzuzeigen.

Eine kleinere Erweiterung, die aus dem Adventsschwimmen hervorging, wäre es, die Raspberry Pis mit LEDs für den Akkustand, Status und zur Bestätigung der Zeitnahme auszustatten. Vor allem eine Rückmeldung zur Zeitnahme würde die Fehlerrate bei den Zeitnehmern deutlich senken, da es häufig der Fall war, dass doppelt oder gar nicht gedrückt wurde, weil sich die Zeitnehmer nicht sicher waren ob sie die Zeit wirklich gestoppt hatten.

Eine weitere Möglichkeit ist es die Raspberry Pis und Drücker durch Smartphones zu ersetzen. Diese bieten den Vorteil, dass nur noch ein kompaktes Gerät für die Zeitmessung benötigt wird, welches viele Funktionalitäten (Monitor, Akkustand, …) und Software zu Verfügung stellt, die sich leicht erweitern lässt. Nachteil wäre es, dass die Smartphones mit einer speziellen Schutzhülle gegen Wasser geschützt werden müssten und sichergestellt werden müsste, dass die Bedienung des Touchscreens nicht durch Spritzwasser beeinträchtigt wird.

9.1.2 Zielrichter-App

Die Zielrichter-App funktioniert grundsätzlich sehr gut und erfüllt seine Aufgabe im Gesamtsystem. Allerdings gibt es hier auch noch Verbesserungs- und Optimierungsmöglichkeiten. Anhand von Überlegungen aus der Projektgruppe und dem Testeinsatz beim Adventsschwimmen, wurden einige Punkte zur Verbesserung und Optimierung gefunden. Einige von diesen Punkten wurden auch von der Zielrichtergruppe umgesetzt, aber nicht alle. Die hier aufgeführten Punkte sind ein Ausblick auf weitere Entwicklungsmöglichkeiten, für neue Projektgruppen, die sich mit diesem Thema auseinandersetzen.

Die momentan benutzen Smartphones erfüllen zwar ihren Zweck, aber gerade für den Einsatz im Schwimmbad, wären hier wasserdichte Smartphones angebracht. Dadurch könnten Unfälle mit Spritzwasser oder anderen Flüssigkeiten vorgebeugt werden. Allerdings ist hier der höhere Preis der Smartphones zu berücksichtigen.

Damit die Smartphones während des Wettkampfes nicht vom Zielrichter festgehalten werden müssen, könnte eine Handyhalterung am Stuhl Abhilfe schaffen. So könnte das Handy auch nicht herunterfallen und kaputt gehen und die Zielrichter könnten entspannter ihren Job durchführen.

Softwaretechnisch wurde sich von den Zielrichtern gewünscht, dass die Positionsreihenfolge, auch durch Tippen auf die Bahn, auf die nächste freie Position springt. Somit könnten die Zielrichter wesentlich schneller ihre Ergebnisse in der App eintragen. Dies ist gerade dann von Vorteil, wenn die Intensität des Schwimmwettkampfes sehr hoch ist und die Zielrichter wenig Zeit haben.

Da so ein Schwimmwettkampf einen ganzen Tag dauern kann und die Smartphones im Dauerbetrieb sind, sind die Akkus ein mögliches Problem. Auch wenn während des Adventsschwimmens kein Handy unter 50% rutschte, ist dies trotzdem ein Risikofaktor. Hier wäre eine Powerbank eine mögliche Lösung. Auch Steckdosen in der Nähe mit entsprechenden Ladekabeln, wären eine Möglichkeit. Die

Zielrichter müssten dadurch keine Zwangspause machen und der Betrieb des Wettkampfes würde nicht aufgehalten werden.

Damit die nachfolgenden Gruppen die Vorteile des Test-Driven-Developments effektiver nutzen können, sollten die UI-Tests auf dem GitLab-Server eingerichtet werden. Zudem sollte die Ursache gesucht werden, warum die aktuellen UI-Tests auch lokal nicht alle erfolgreich sind. Das Gesamtsystem funktioniert und es sind keine offensichtlichen Fehler der Zielrichter-App bekannt oder während der letzten Tests, auch nach dem Schwimmkampf, aufgefallen. Die Fehlerursache könnte daher nicht unbedingt ein Fehler im Quellcode sein, sondern es könnte sich auch um einen Fehler im Design, um einen Fehler im Zusammenspiel zwischen der Applikation und den Testklassen oder nur um einen Fehler innerhalb der Testklassen handeln.

9.1.3 Kontrollprogramm

Wie bereits im Abschnitt 8.1.3 Kontrollprogramm geschildert wurde, wäre es wünschenswert, einen Drucker zu ergänzen, der zum einen den Vorteil hätte, dass Zeiten für die Siegerehrung sofort zu verlesen wären. Zum anderen kann man ebenfalls Daten von den Zielrichtern mit ausdrucken und hätte somit ein durchlaufendes Backup-Medium, falls Daten verloren gehen sollten.

Man sollte sich unbedingt um die Sicherheit der MQTT-Kommunikation kümmern. Momentan wird, bis auf kleine Anpassungen, die Standardkonfiguration verwendet. Die Verbindung wird zurzeit nicht verschlüsselt und Benutzer müssen sich nicht authentifizieren. Aufgrund des Umfangs hat man beschlossen, dieses Thema erstmal hintenanzustellen und die Priorität auf die funktionalen Anforderungen gelegt.

Überdenken kann man nochmal das Verfahren der Überprüfung, ob alle Geräte erreichbar sind. Von Vorteil wäre es sicher auch, wenn der Benutzer durch einen Popup und/oder akustisches Signalton auf ein Verlieren der Verbindung aufmerksam gemacht wird.

Ebenfalls kann ein Algorithmus ergänzt werden, der erkennt, wenn eine Zwischenzeit fehlt und die Zeile markiert und/oder ein akustischen Signalton abgibt, um den Benutzer zu signalisieren, dass möglicherweise der Zeitnehmer nicht gedrückt oder das Gerät keine Zeit übermittelt hat.

Momentan ist das Layout auf ein 10,8”-Monitor optimiert. Das Design ist nicht komplett responsiv, sodass einige Teile bei kleineren Displays verschwinden können. Bei größeren Displays wäre es von Vorteil, wenn die Schrift auch größer werden würde, um nicht zu große leere Lücke zu erhalten.

9.1.4 Auswerterprogramm

In einer zukünftigen Iteration können noch einige Dinge verbessert werden. Ein optimaler Zielrichter-Algorithmus konnte bis zum Adventsschwimmen nicht mehr fertiggestellt werden. Der derzeitige Algorithmus funktioniert in dem meisten Fällen, deckt aber spezielle Corner-Cases nicht ab. Für diese Corner-Cases wurde die Methode fixZielrichterResults() geschrieben, diese wurde aber kurz vor dem Wettkampf nach Rücksprache mit Herrn Prof. Dr. Rabe wieder entfernt. Auch kann es noch vorkommen, dass der Algorithmus eine Bahn auf zwei Platzierungen gleichzeitig setzt. Ein Problem tritt auf, wenn ein Zielrichter eine Bahn mithilfe der App nicht auf den ersten, sondern den zweiten Platz schiebt. Den ersten Platz lässt er frei. Laut Anforderungen ist die Abgabe des Zielrichters nur eine relative Positionierung, d.h. die Bahn, welche er auf Platz zwei geschoben hat, wird als sein erster Platz gewertet. Hat jedoch ein anderer Zielrichter Platz 1 mit einer Bahn belegt, dürfen diese Angaben nicht gleich bewertet werden. Dies konnte nicht zufriedenstellend in einem Algorithmus umgesetzt werden.

Eine weiterreichende Verbesserung wäre es, die Bahnen in der Spalte ZR.amtlich per Drag & Drop zu verschieben, wie es bei vielen modernen Programmen üblich ist. Dieses Feature wurde aus Zeit- und Erfahrungsmangel nicht umgesetzt.

Zurzeit wird für Platzierungen, welchen keine Bahn zugeordnet wurde, weil es z.B. nicht genügend Schwimmer gab, um alle Bahnen zu belegen, ein „n.E.“ eingetragen. Dies steht für „Nicht erschienen“. Besser wäre, wenn dort unterschieden werden kann ob ein Schwimmer nicht erschienen ist, oder die Bahn auch laut Meldeergebnis gar nicht belegt werden sollte. Dies war nicht möglich, da das Auswerter Programm kein Meldeergebnis erhalten hat und somit nicht wissen konnte, wie viele Schwimmer vorgesehen waren.

In Übersicht aller Zwischenzeiten können bisher nur Zeiten gelöscht werden. Sehr nützlich wäre es, wenn sich dort Zeiten verschieben lassen und neue hinzugefügt werden könnten. Bisher ist ein Beheben von Fehlern bei den Zeiten im Auswerter-Programm nur auf das Löschen einzelner Zeiten begrenzt.

Die Anzahl der Bahnen könnte erhöht werden oder im Idealfall sogar frei konfigurierbar sein.

Eine farbliche Markierung als Signal, ob es Ungereimtheiten gibt, findet derzeit unter der Tabelle statt. Besser wäre es, wenn direkt die betroffenen Zellen markiert werden würden.

Das UI-Design nutzt noch das standardmäßige Aussehen von JavaFX, während das Kontrollprogramm bereits ein eigenes Design verwendet. Um ein durchgehendes Look & Feel zu erreichen, kann das UI entsprechend angepasst werden.

9.1.3 Lenex

Die Lenex-Funktionalität ist schon ziemlich fortgeschritten. Sowohl der Import der Lenex-Datei vom Kontrollrechner als auch der Rückimport in EasyWk funktioniert ohne Probleme vor Projektabgabe. Jedoch kann der Rückimport in Richtung Konformität sich weiterentwickeln. Derzeitig kann nur Lauf für Lauf importiert werden, was folglich etwas mühsam sein kann. Die Überlegung ist nun, den Rückimport von mehreren Läufen oder von ganzen Wettkämpfen mit einem Klick zu ermöglichen. Ob es überhaupt ohne weiteres möglich ist, sei an dieser Stelle hingestellt. Ggf. ist eine Absprache mit dem EasyWk-Entwickler Björn Stickan erforderlich.

Des Weiteren ist das Problem, dass keine Zielrichterentscheide mitgegeben werden können. Über das ALGE-Austauschverfahren in EasyWk werden Hardwarezeiten erwartet. Problem an dieser Stelle: Die Zeiten, die mittels dem FileChangeListener mitgegeben werden, sind bereits amtlich. Jedoch fehlt die Möglichkeit bei gleichen amtlichen Zeiten in einem Lauf Zielrichterentscheids-Informationen mit an EasyWK zu übermitteln. Auch hier ist eine Absprache mit dem EasyWk-Entwickler Björn Stickan erforderlich.

Wie in **Fehler! Verweisquelle konnte nicht gefunden werden.** beschrieben wird ein DNF (did not finished) in EasyWk geschrieben, sofern ein Schwimmer seinen Lauf nicht antreten konnte. Um genauere Gründe zu spezifizieren ist es möglich weitere Status anzugeben. Hierbei steht zur Debatte, ob ein DNS (did not start) doch eher sinngemäß ist. EasyWK ist fähig folgende Status zu interpretieren:

- EXH: exhibition swim.
- DSQ: athlete/relay disqualified.
- DNS: athlete/relay did not start (no reason given or to late withdrawl).
- DNF: athlete/relay did not finish.
- SICK: athlete is sick.
- WDR: athlete/relay was withdrawn (on time).

10 Fazit

Das Projekt konnte rechtzeitig zum Adventsschwimmen 2018 fertig gestellt werden und dort auch erfolgreich eingesetzt werden. Jedoch ist es noch kein vertriebsbereites Produkt. Das liegt zum einen an dem relativ großen Funktionsumfang, den das Produkt enthalten sollte, sowie an der Unerfahrenheit fast aller Gruppenmitglieder. Aufgrund dieser Unerfahrenheit ließen sich nur schwierig realistische Termine für Meilensteine planen. Große Teile des Systems wurden außerdem in der Planung nicht gut durchdacht, und mussten später aufwändig geändert werden (Siehe 6.4.2. Verwurf der Datenbank). Den Meilensteinen wurde aufgrund dessen nur wenig Aufmerksamkeit während des Projektverlaufs geschenkt.

Das Projekt war außerdem sehr ambitioniert, dafür dass es ein Studierendenprojekt war. Es war so umfangreich, dass es gegen Ende zu einem großen Problem wurde, einen Test des Gesamtsystems durchzuführen. So ein Test ließ sich auch nicht virtuell und automatisiert durchführen, da so viele verschiedene Hardwarekomponenten benötigt wurden.

Das Projekt sollte außerdem mit Test-Driven-Development entwickelt werden. In der Theorie sollte dieses Vorgehen den Entwicklungsprozess vereinfachen, indem Bugs schon früh erkannt werden. Praktisch wurde TDD allerdings nur wenig eingesetzt. Die Gründe dafür waren vielfältig. Einige hatten keine Zeit sich zusätzlich zu dem gewählten Framework, in dem das Projekt programmiert wurde, auch noch in ein Test-Framework einzuarbeiten. Andere wussten ganz einfach nicht wie TDD angegangen werden soll. Beispielsweise waren Tests für die grafische Oberfläche nicht leicht zu schreiben, stellten aber aus Sicht der Gruppe auch keinen Mehrwert dar. Stattdessen wurde oft einfach durch „rumklicken“ getestet.

Die Planungsphase hätte besser genutzt werden können. Die Anforderungen an das Produkt waren lange unklar und haben sich auch während der Entwicklung noch oft geändert. Besonderer Augenmerk ist hier auf den Kontrollrechner zu richten, dem in der Planungsphase nur zwei Personen zugeteilt wurden, da der Funktionsumfang zu diesem Zeitpunkt noch sehr begrenzt war. Im Laufe des Projekts wurden immer wieder neue Anforderungen gestellt, sodass das Kontrollprogramm nun mit Abstand das größte Teilprojekt ist. Besonders hier wurden die Tests vernachlässigt - viel wichtiger war es, die immer mehr werdenden Funktionen mit nur zwei Entwicklern umzusetzen, damit der Zeitplan eingehalten werden und das System zum Adventsschwimmen eingesetzt werden konnte.

Die Kommunikation innerhalb der Projektgruppe hat allerdings sehr gut funktioniert, sodass keine Gruppe ohne Rücksicht auf Kompatibilität programmiert hat. Eine gute Entscheidung war außerdem, dass alle Kleingruppen die Sprache Java benutzt haben. So konnten Klassen für die Kommunikation einfach und schnell ausgetauscht werden. Lediglich die Gruppe, welche die RaspberryPis bearbeitete, hat aus Geschwindigkeitsgründen die Sprache C verwendet. Hinsichtlich des Kompetenzgewinns war das Projekt für alle Teilnehmer ein großer Erfolg.

Anhang

A Installations- und Bedienungsanleitung

A.1 Zeitmesssystem

A.1.1 Betriebssystem Konfiguration

Zuerst muss das Betriebssystem von https://www.raspberrypi.org/downloads/raspbian/ gedownloadet werden. Es wird die Lite Version empfohlen, da diese ohne GUI oder weitere Software daherkommt. Das Betriebssystem ist für alle Modelle des Raspberry Pis identisch.

Von einem Windows-System kann diese Image Datei nun mit einem Programm wie Etcher (https://www.balena.io/etcher/) oder Win32 Disk Imager (https://sourceforge.net/projects/win32diskimager/) auf eine SD-Karte geschrieben werden.

Die Konfiguration kann entweder per SSH geschehen oder es müssen Tastatur und Monitor an den Pi angeschlossen werden. Hierbei ist darauf zu achten, dass der RPi Zero W lediglich einen Micro-USB und einen Mini-HDMI Anschluss bietet.

Entscheidet man sich für die Konfiguration über SSH muss in der Boot-Partition des erstellten Mediums eine leere Datei mit dem Namen *ssh* erstellt werden. Ebenfalls muss eine Datei mit dem Namen *wpa_supplicant.conf* mit folgendem Inhalt erstellt werden:

```
network={
ssid="Schwimmwettkampf_AP"
psk="tddschwimmwettkampf"
proto=RSN
key_mgmt=WPA-PSK
pairwise=CCMP
auth_alg=OPEN
}
```

Abbildung 48 wpa_supplicant.conf

ssid und *psk* müssen auf ein vorhandenes Netzwerk angepasst werden, in dem sich ebenfalls unser Gerät befindet, mit dem die Konfiguration vorgenommen werden soll. Ebenfalls ist darauf zu achten, dass ein Zugriff auf den Router möglich sein sollte, um auf die IP-Adresse des verwendeten Raspberry Pi schließen zu können.

Nach dem ersten Boot wird der Inhalt dieser Datei automatisch nach *etc/wpa_supplicant/wpa_supplicant.conf* kopiert.

Unter Windows wird erneut zusätzliche Software, wie beispielsweise Putty (https://www.putty.org/) benötigt, um eine SSH Verbindung aufzubauen.

Jetzt kann die SD-Karte in den Raspberry Pi gesteckt werden.

Der Standard Login geschieht über den Benutzer „Pi“ mit dem Passwort „Raspberry“. Bei der Eingabe ist darauf zu achten, dass noch ein englisches Tastaturlayout aktiviert ist. Mit dem Befehl *sudo raspi-config* kommt man in ein grafisches Optionsmenü. Hier kann man unter dem Punkt *„Internationalization --> Keyboard setup“* das gewünschte Layout auswählen. Um die Änderung zu übernehmen muss ein Reboot durchgeführt werden.

A.1.2 Echtzeit Betriebssystem Update

Da die Raspberry Pi’s unter Echtzeitanforderungen betrieben werden sollten, musste der Betriebssystemkernel zusätzlich geupdatet werden, damit niedrige Latenzzeiten erreicht werden können. Die folgenden Schritte 1 bis 4 sollten für die schnelle Kompilierung auf einem externen

Rechner (Linux) durchgeführt werden, außerdem kann der Kernel nach erfolgreichem Kompilieren auf die anderen Raspberry Pi's kopiert werden.

1. Laden Sie die aktuelle Version des Kernels (rpi-4.14.y) von GitHub herunter.

```
~/rpi-kernel$ git clone https://github.com/raspberrypi/linux.git
```

```
~/rpi-kernel$ cd linux
```

```
~/rpi-kernel/linux$ git checkout rpi-4.14.y-rt
```

2. Konfigurieren Sie die Tools für die Cross-Compilierung. Hier sollte genauestens auf die richtige Eingabe der Befehle geachtet werden, da die Fehler erst beim Kompilieren auftreten.

```
~/rpi-kernel$ git clone https://github.com/raspberrypi/tools.git
```

```
~/rpi-kernel$ export ARCH=arm
```

```
~/rpi-kernel$ export CROSS_COMPILE=~/rpi-kernel/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux-gnueabihf-
```

```
~/rpi-kernel$ export INSTALL_MOD_PATH=~/rpi-kernel/rt-kernel
```

```
~/rpi-kernel$ export INSTALL_DTBS_PATH=~/rpi-kernel/rt-kernel
```

3. Konfigurieren Sie den Kernel und editieren Sie die .config Datei für die Fully Preemptiable Kernel und High Resolution Timer Support Option.

```
~/rpi-kernel/linux$ export KERNEL=kernel
```

```
~/rpi-kernel/linux$ make bcmrpi_defconfig
```

```
~/rpi-kernel/linux$ nano .config
```

HIGH_RES_TIMERS=y

CONFIG_PREEMPT_RT_FULL=y

CONFIG_HZ_1000=y

CONFIG_HZ=1000

4. Kompilieren Sie den Kernel, über die Option -j können Sie einstellen wie viele CPU-Kerne zum Kompilieren genutzt werden sollen.

```
~/rpi-kernel/linux$ make -j4 zImage
```

```
~/rpi-kernel/linux$ make -j4 modules
```

```
~/rpi-kernel/linux$ make -j4 dtbs
```

```
~/rpi-kernel/linux$ make -j4 modules_install
```

```
~/rpi-kernel/linux$ make -j4 dtbs_install
```

5. Installieren Sie den Kernel auf dem Raspberry Pi.

```
~ / sudo -s
```

```
~ / sudo export KERNEL=kernel7
```

~ / ./scripts/mkknlimg ./arch/arm/boot/zImage /boot/$KERNEL.img

~ / cp -pv ./arch/arm/boot/dts/*.dtb /boot

~ / cp -rpv ./arch/arm/boot/dts/overlays /boot

~ / exit

6. Öffnen Sie die cmdline.txt Textdatei und fügen sie die zwei unteren Zeilen hinzu.

~/ sudo nano /boot/config.txt

dwc_otg.fiq_enable=0

dwc_otg.fiq_fsm_enable=0

7.Nach der Installation starten Sie den Raspberry Pi neu und überprüfen Sie die Kernelversion, um festzustellen, dass der Kernel erfolgreich geupdatet wurde.

~/ sudo reboot

~/ uname -a

A.1.3 MQTT Installation

Wir benutzen den Paho MQTT C Client (https://github.com/eclipse/paho.mqtt.c).

Diese Library benötigt das OpenSSL Entwickler Package. Installiert werden kann dieses mit dem Befehl *sudo apt-get install libssl-dev*.

Für den Download wird git benötigt. Dieses kann installiert werden mit dem Befehl *sudo apt-get install git*.

Daraufhin kann die MQTT-Library mit dem Befehl *git clone https://github.com/eclipse/paho.mqtt.c* gedownloadet werden. Anschließend kann in den Ordner gewechselt und mit den Befehlen *sudo make* und *sudo make install* die Installation abgeschlossen werden.

A.1.4 wiringPi-Bibliothek + Programm

Die wiringPi-Bibliothek wird für die Abfrage der Raspberry Pi GPIOs benötigt. Mit *gpio –v* kann überprüft werden, ob wiringPi schon auf dem System vorhanden ist.

Für den Download wird git benötigt. Dieses kann installiert werden mit dem Befehl *sudo apt-get install git*.

Über den Befehl *sudo git clone git://git.drogon.net/wiringPi* kann die Bibliothek heruntergeladen werden und im wiringPi Ordner mit *sudo ./build* kompiliert werden.

Das Programm kann über den Befehl *sudo git clone https://github.com/Kenoooo/Pi-Druecker* heruntergeladen werden. Später müssen innerhalb des Programms die Topics, sowie die ClientID an jedes Gerät einzeln angepasst werden. Kompiliert werden kann das Programm mithilfe des Befehls *make clean*.

A.1.5 PTP Konfiguration

Zur Zeitsynchronisation der Raspberry Pi's wurde ptpd2 Dienst für Debian verwendet. Die Installation erfolgt über *sudo apt-get install ptpd,* des weiterem muss die Konfigurationdatei *ptpd2.conf* unter /etc/ptpd2/ptpd2.conf entsprechend Abbildung angepasst werden. Über *sudo service ptpd start/stop* kann der Dienst gestartet oder gestoppt werden. Zum Testen von der PTP-Verbindung kann die Logdatei /var/log/ptpd2.log genutzt werden.

```
ptpengine:interface=wlan0
ptpengine:domain=42
ptpengine:preset=masterslave
ptpengine:ip_mode=multicast
ptpengine:use_libpcap=n
clock:drift_handling=file
global:log_file=/var/log/ptpd2.log
global:log_file_max_size=100
global:log_file_max_files=2
global:log_status=y
global:cpuaffinity_cpucore=0
;
```

Abbildung 49 ptpd2.conf

Ab diesem Punkt kann die SD-Karte mit einem Programm wie Win32 Disk Imager (https://sourceforge.net/projects/win32diskimager/) gesichert werden und danach auf die gewünschte Anzahl an Karten kopiert werden.

A.1.6 NTP Konfiguration

NTP-Client

Auf einem der Geräte sollte nun ein NTP-Client installiert werden. Dieser erhält die "korrekte" Zeit und informiert die restlichen Geräte über das PTP-Netz.

Der NTP-Client wird mit dem Befehl *sudo apt-get install ntp* installiert. Stoppen und deaktivieren Sie die Systemzeitsynchronisation der Raspberry Pi's über *systemct1 stop systemd-timesyncd* und *systemct1 disable systemd-timesyncd.* Für die Konfiguration muss in der */etc/ntp.conf* nur der Punkt #server ntp.your-provider.example durch die Kontrollrechner IP-Adresse (192.168.213.66) ersetzt und auskommentiert werden.

```
# You do need to talk to an NTP server or two (or three).
#server ntp.your-provider.example
server 192.168.213.66
```

Abbildung 50 /etc/ntp.conf

NTP-Server

Der NTP-Server wird auf dem Kontrollrechner unter Windows eingerichtet.

1. Geben Sie in der Windowssuche „Verwaltung" ein, suchen Sie unter Dienste nach „Windows-Zeitgeber" und stoppen Sie ihn, falls dieser gestartet ist. (Schließen Sie diese Fenster nicht)

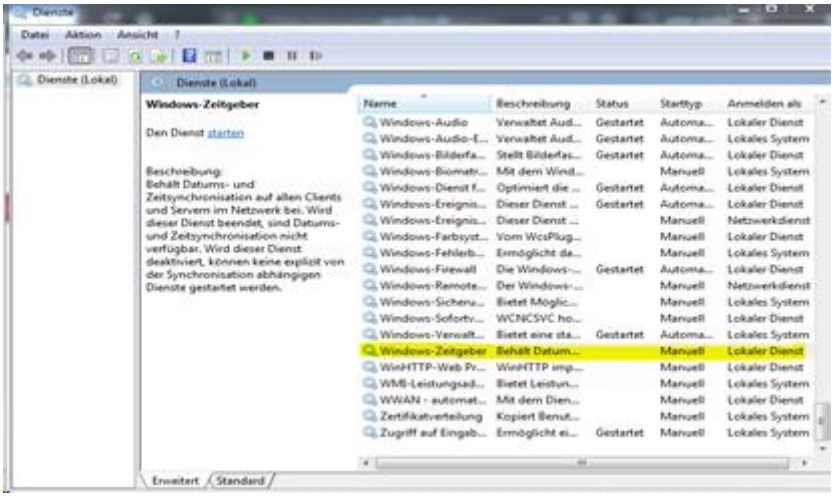


Abbildung 51 Dienst Windows-Zeitgeber

2. Suchen Sie nun erneut in die Windowssuche nach dem Programm „regedit.exe“ und führen Sie es aus.
3. Öffnen Sie HKEY_LOCAL_MACHINE > SYSTEM > CurrentControlSet > services > W32Time > TimeProviders > NtpServer > Enabled und setzen Sie den Wert auf 1.
4. Öffnen Sie HKEY_LOCAL_MACHINE > SYSTEM > CurrentControlSet > services > W32Time > Config > AnnounceFlags und setzen Sie den Wert auf 5.

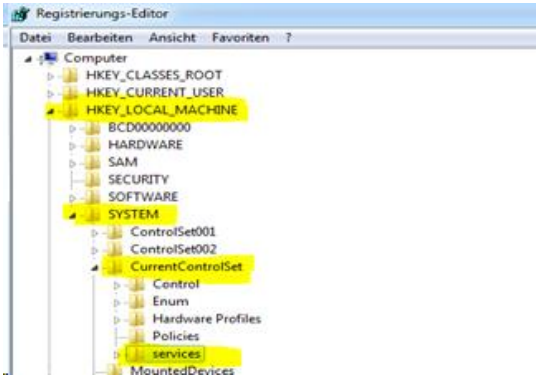


Abbildung 52 regedit.exe

5. Gehen Sie zurück zum „Windows-Zeitgeber“ aus Schritt 1 und stellen Sie den Starttyp auf „automatisch“.

A.1.7 Einrichtung des Access-Points

Auf einem der Geräte sollte nun ein Access-Point eingerichtet werden. Zuerst müssen die zwei Libraries *dnsmasq* und *hostapd* installiert werden. Dies geschieht über den Befehl *sudo apt-get install dnsmasq hostapd*. Anschließend sollten beide Services zuerst gestoppt werden, mit den Befehlen *sudo systemctl stop dnsmasq* und *sudo systemctl stop hostapd*.

Als erstes sollte jetzt /etc/network/interfaces wie folgt angepasst werden:

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface wlan1 inet manual
```

Abbildung 53 /etc/network/interfaces

Als nächstes muss auf dem zu verwendenden Netzwerkadapter eine statische IP festgelegt werden. Diese kann in der *dhcpcd.conf* im Ordner */etc/* konfiguriert werden. Unsere Datei sah danach wie folgt aus:

```
interface wlan1
        static ip_address=192.168.213.1/24
        nohook wpa_supplicant
```

Abbildung 54 /etc/dhcpcd.conf

In der *interface* Zeile ist der zu verwendende Netzwerkadapter anzugeben. In der darauffolgenden Zeile kann die IP beliebig festgelegt werden. Die Adresse eines Routers ist typischerweise die erste im jeweiligen Subnetz.

Der dhcpcd Service kann jetzt mit dem Befehl *sudo service dhcpcd restart* neugestartet werden.

Jetzt kann das Dynamic Host Configuration Protocol (DHCP) konfiguriert werden. Die zu bearbeitende Datei befindet sich in */etc/dnsmasq.conf*. Sie sollte wie folgt aussehen:

```
interface=wlan1
dhcp-range=192.168.213.128,192.168.213.160,255.255.255.0,24h
```

Abbildung 55 /etc/dnsmasq.conf

Auch hier ist *interface* erneut auf den verwendeten Adapter anzupassen. Die folgende Zeile bedeutet, dass IP Adressen im Bereich von 192.168.213.128 bis 192.168.213.160 vergeben werden, mit der Netzmaske 255.255.255.0. Die Lease-Zeit beträgt 24 Stunden.

Daraufhin kann die Host-Software konfiguriert werden. Hierbei ist die Datei */etc/hostapd/hostapd.conf* zu bearbeiten.

```
interface=wlan1
driver=nl80211
ssid=Schwimmwettkampf_AP
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=tddschwimmwettkampf
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Abbildung 56 /etc/hostapd/hostapd.conf

Die Zeilen *ssid* und *wpa_passphrase* können nach Belieben angepasst werden.

Nun muss diese Datei noch in */etc/default/hostapd* referenziert werden, indem die Zeile *DAEMON_CONF=““* zu *DAEMON_CONF=“/etc/hostapd/hostapd.conf“* geändert wird.

```
# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
#       -d   show more debug messages (-dd for even more)
#       -K   include key data in debug messages
#       -t   include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
#
#DAEMON_OPTS=""
```

Abbildung 57 /etc/default/hostapd

Schlussendlich können die Services wieder gestartet werden, mit den Befehlen *sudo systemctl start hostapd* und *sudo systemctl start dnsmasq*.

Jetzt muss noch in */etc/sysctl.*conf die Zeile *net.ipv4.ip_forward=1* einkommentiert werden. Danach sollte der ausgehende Verkehr auf *eth0* maskiert werden, mit dem Befehl *sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE* und anschließend die iptables-Regel mit dem Befehl *sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"* gespeichert werden.

Zuletzt sollte in der Datei */etc/rc.local* die Zeile *iptables-restore < /etc/iptables.ipv4.nat* ergänzt werden, um die vorher definierten Regeln beim nächsten Boot zu installieren. Ebenfalls ist es empfehlenswert in selbiger Datei noch die Zeile *iw dev wlan1 set power_save off* hinzuzufügen. Diese deaktiviert das Power-Management für den Netzwerkadapter. Dies sorgt dafür, dass der Adapter sich nicht nach längerer Untätigkeit selbständig deaktiviert, was zu einem zeitweisen Ausfall des Netzwerks führen kann. Letztendlich sieht */etc/rc.local* also wie folgt aus:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

iw dev wlan1 set power_save off

iptables-restore < /etc/iptables.ipv4.nat

exit 0
```

Abbildung 58 /etc/rc.local

A.1.8 Netzwerkkonfiguration

Dieser Schritt sollte jetzt auf jedem Gerät, außer dem zuvor konfigurierten Access-Point geschehen.

Zuerst muss die *wpa_supplicant.conf*, die sich im Ordner */etc/wpa_supplicant/* befindet, auf unseren neuen Access-Point angepasst werden, beziehungsweise erstmalig angelegt werden, falls die vorherige Konfiguration nicht über SSH geschehen ist. Informationen dazu sind in Abschnitt A.1.1.

Danach kann in die Datei *interfaces* im Ordner */etc/network/* folgender Inhalt geschrieben werden:

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet static
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
address 192.168.213.35
netmask 255.255.255.0
gateway 192.168.213.1
```

Abbildung 59 /etc/network/interfaces

Die Adress-Zeile ist für jeden zu verwendenden Raspberry Pi individuell anzupassen. In unserem Fall wurden Adressen ab 192.168.213.32 aufwärts zugewiesen. Das Gateway muss auf die IP des Gerätes mit Hostapd angepasst werden.

A.1.9 SSH-Verbindung zum Raspberry

Windows

1. Laden Sie sich PuTTY (putty.org) für Windows herunter und öffnen Sie die putty.exe Datei.
2. Geben Sie als Host den Namen des Raspberry Pi‘s oder die IP-Adresse des Ziels ein, als Port nehmen Sie den Standard Port 22.

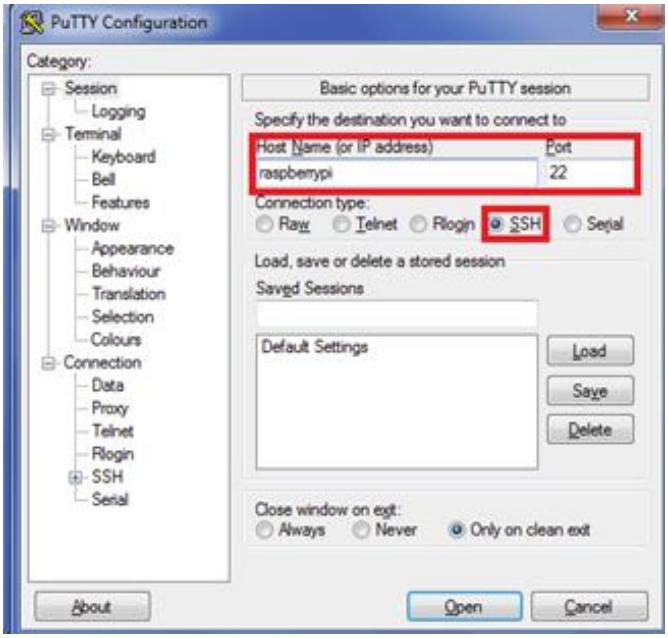


Abbildung 60 SSH Verbindung aufbauen

Linux & Mac

Öffnen Sie das Terminal (Strg + Alt + T) und gegen Sie den Befehl *ssh Benutzer@IP-Adresse -p 22* ins Terminal ein.

A.2 Zielrichter-App

Um die Zielrichterapp des halbautomatischen Systems zur Erfassung, Bearbeitung und Auswertung von Wettkampfdaten für Schwimmwettkämpfe zu benutzen und erfolgreich zu bedienen ist folgendes zu beachten:

Damit die App auf dem Smartphone installiert wird, braucht man eine geeignete Entwicklungsumgebung für Android-Apps. Zu empfehlen ist hier die Umgebung Android Studio. Die App ist als Projekt für Android Studio im Gitlab der Hochschule vorhanden und kann über die URL https://gitlab.repo.digitech.hs-emden-leer.de/rabe/projektgruppe-ws18/zielrichter-app/zielrichter-app heruntergeladen werden.

Nachdem das Smartphone in den Entwicklermodus versetzt und an den PC angeschlossen wurde, wählt man über das Klicken des Buttons „Run-App“ das entsprechende Device aus. Dadurch wird die App auf dem Smartphone installiert oder gegebenenfalls auf einem Android-Emulator ausgeführt, sofern man diesen ausgewählt hat. Ein Download über den Google-Playstore wurde nicht vorgesehen.

Ist die App erfolgreich installiert worden kann man diese über Anklicken des App-Symbols (Pfeife) auf dem Smartphone starten. Nach dem Start kommt man auf einen Login-Screen. Um Sich einzuloggen braucht man einen Code, der vom Kontrollrechner generiert wird. Dieser muss dort erfragt werden. Nach dem erfolgreichen Einloggen kommt man auf einen Platzierungs-Screen. Hier kann man die Bahnen je nach angekommener Reihenfolge auf die Plätze per Drag and Drop ziehen. Ist dies geschehen drückt man den Button „Absenden“. Danach erfolgt eine Übersicht zur Kontrolle der gezogenen Bahnplatzierungen. Ist man hiermit zufrieden, bestätigt man dies mit drücken des Buttons „Absenden“. Nach dem Versenden der Ergebnisse landet man auf der Abgabenübersicht. Hier sieht man nun einen grünen Haken bei sich selbst und den Status der Abgaben der anderen Zielrichter. Hier bleibt man solange bis alle Zielrichter abgegeben haben, der Master den Lauf beendet oder der Kontrollrechner den Lauf für abgeschlossen erklärt.

Zusätzlich kann jeder Zielrichter im Menü den Zielrichterstatus seiner Zielrichterkollegen abrufen. Zudem ist es über das Menü möglich sich abzumelden. Ist man der „Master“ der Zielrichter, so kann man im Menü auch eine Wettkampfauswahl öffnen. Hierrüber kann der Master einen neuen Lauf seiner Wahl auswählen und starten.

A.3 Kontrollprogramm

Damit die Inbetriebnahme des Kontrollprogramms erfolgen kann, muss das Projekt zuerst aus GitLab heruntergeladen werden. Das Projekt sollte danach in der Entwicklungsumgebung importiert und als Maven konvertiert werden (siehe dazu A.4.1 Projekt öffnen und kompilieren). Danach muss der Branch auf kontroll-app gewechselt werden. Da die IP-Adresse des MQTT-Brokers in den Raspberry Pis der Zeitnehmer fest programmiert ist, muss die IP-Adresse statisch im Rechner gesetzt werden. Die IP-Adresse muss ebenfalls dem Auswerter-PC mitgeteilt werden. Zur Zeitsynchronisierung der Zeitnehmer, muss der PC des Kontrollprogramms als NTP-Server aktiviert werden (Siehe dazu A.1.6 NTP Konfiguration). Zum Starten sollte Spring Boot vorher installiert sein. Im Anschluss kann das Programm unter Spring gestartet werden und das Startfenster öffnet sich.

Im Schritt 1 des Startvorgangs muss entschieden werden, ob eine neue Veranstaltung erstellt wird oder eine vorherige wiederhergestellt werden soll, sofern die Backup-Dateien für diese existieren. Klickt man auf “Weiter”, so gelangt man zum Schritt 2, in dem man die Nachlaufzeit bei einem Überkopf-Start zwischen 0 und 30 Sekunden festlegen kann, die Wettkampfdaten als .lef-Datei einlesen und die Zielrichter verwalten kann. Es können bis maximal 9 Zielrichter angelegt werden. Bei Klick auf “Zielrichter hinzufügen” werden immer so viele Zielrichter hinzugefügt, dass die Anzahl an Zielrichtern ungerade ist.

Durch Klick auf “Start” öffnet sich das Dashboard, auf dem alle Daten des Wettkampfes übersichtlich dargestellt sind. Rechts oben kann man nun ermitteln, welche Geräte mit dem Netzwerk verbunden sind und ihren Status korrekt senden.

Testweise kann man auch Läufe starten und über die Fehlstart-Funktion wieder zurücksetzen. Um Läufe zu verschieben, muss man die Wettkampfübersicht öffnen und dort in der Liste per Drag and Drop den Lauf an die gewünschte Stelle schieben.

Falls bereits gelaufene Läufe in der Wettkampfübersicht vorhanden sind, die allerdings nicht vom Kontrollrechner betreut wurden, so kann man in der Wettkampfübersicht diese anklicken und in den Details auf der linken Seite unter dem Punkt Zeitnehmer durch Klick auf “beenden” den Lauf als beendet markieren. Nun wird dieser Lauf beim nächsten Startsignal nicht mehr berücksichtigt.

Weitere detaillierte Beschreibungen der einzelnen Funktionalitäten und Benutzeroberflächen können unter den Punkt 6.7 Kontrollprogramm nachgelesen werden.

A.4 Auswertungsprogramm

In den folgenden zwei Abschnitten wird beschrieben, wie das aktuelle Auswertungsprogramm auf dem eigenen PC aufgerufen und im Anschluss daran für die Auswertung von Wettkämpfen verwendet werden kann.

A.4.1 Projekt öffnen und kompilieren

Das Auswerterprogramm wurde in der Entwicklungsumgebung Eclipse geschrieben. Ob das Projekt auch ohne weiteren Konfigurationsaufwand in anderen Entwicklungsumgebungen funktioniert, wurde nicht getestet. Eclipse benötigt Spring Tools in der Version 3. Das Addon e(fx)clipse wird ebenfalls verwendet, ist allerdings nicht zwingend notwendig. Dies ermöglicht eine angenehmere Entwicklung von JavaFX Komponenten mit Eclipse. Zum Zeitpunkt dieser Dokumentation lassen sich diese über den Eclipse Marketplace installieren. In Eclipse muss das, im gitlab der Hochschule zu findende, Projekt über File > Import… > Project from Git > Clone URI geladen werden. Da sich keine .project-Datei in der Ordnerstruktur befindet, muss „Import as general Project“ gewählt werden. Nachdem das Projekt importiert wurde, muss es noch zu einem Maven-Projekt konvertiert werden. Dies macht man mit einem Rechtsklick auf das Projekt > Configure > Convert to Maven Project. Das Projekt kann nun bearbeitet werden. Außerdem ist drauf zu achten ebenfalls das Projekt „database-parser“ zu klonen. Dies wird zwingend benötigt, da sonst nicht alle Abhängigkeiten erfüllt sind. Der database-parser muss lediglich geklont werden. Weitere Schritte sind nicht notwendig. Bei der Weiterentwicklung der App ist zu beachten, dass die „MainLayout.fxml”-Datei eine händisch hinzugefügte „ArrayList” enthält. Da im Normalfall Änderungen des Designs/Layout mit dem SceneBuilder getätigt werden, muss diese Zeile mit der ArrayList nach jeder Änderung manuell wieder in den Quellcode eingefügt werden. Das liegt daran, dass der SceneBuilder die ArrayList nicht kennt und alle direkt am Quellcode vorgenommenen Änderungen nach der Benutzung dieses Tools wieder verwirft.

Als JAR-Datei kompilieren lässt sich das Projekt auf zwei Weisen: direkt in Eclipse oder über ein Terminal. Um das Projekt aus Eclipse zu kompilieren: Rechtsklick auf die Datei pom.xml > Run as > Run Configuration: In das Feld goals „package“ eintragen und den Haken bei „Skip Test“ setzen. Falls alle Tests erfolgreich sind, muss dieser Haken nicht gesetzt werden. Nach einem Klick auf Run sollte sich nun im Ordner target innerhalb des Projekt-Ordners eine ausführbare JAR-Datei befinden.

Um das Projekt über bash zu kompilieren, muss lediglich in das Wurzelverzeichnis des Projekts navigiert werden und der Befehl „mvn package -DSkipTests“ ausgeführt werden. Nun sollte sich die ausführbare JAR-Datei im target Ordner befinden.

A.4.2 Bedienung

Zuerst sollte sichergestellt werden, dass sich der Computer, auf dem das Programm ausgeführt wird, im selben Netzwerk wie der Kontrollrechner befindet. Dieser muss erreichbar sein, damit die Kommunikation funktioniert. Standardmäßig ist die Adresse des Kontrollrechners im Programm mit „iot.eclipse.org“ hinterlegt, das ist ein öffentlich zugänglicher Testserver für MQTT. Sobald das Programm gestartet ist, wird versucht eine Verbindung zum Kontrollrechner aufzubauen. Gelingt dies nicht, erscheint eine Fehlermeldung und die Verbindung sollte geprüft werden.

Sobald das Programm korrekt eingerichtet ist, öffnet sich folgendes Fenster.

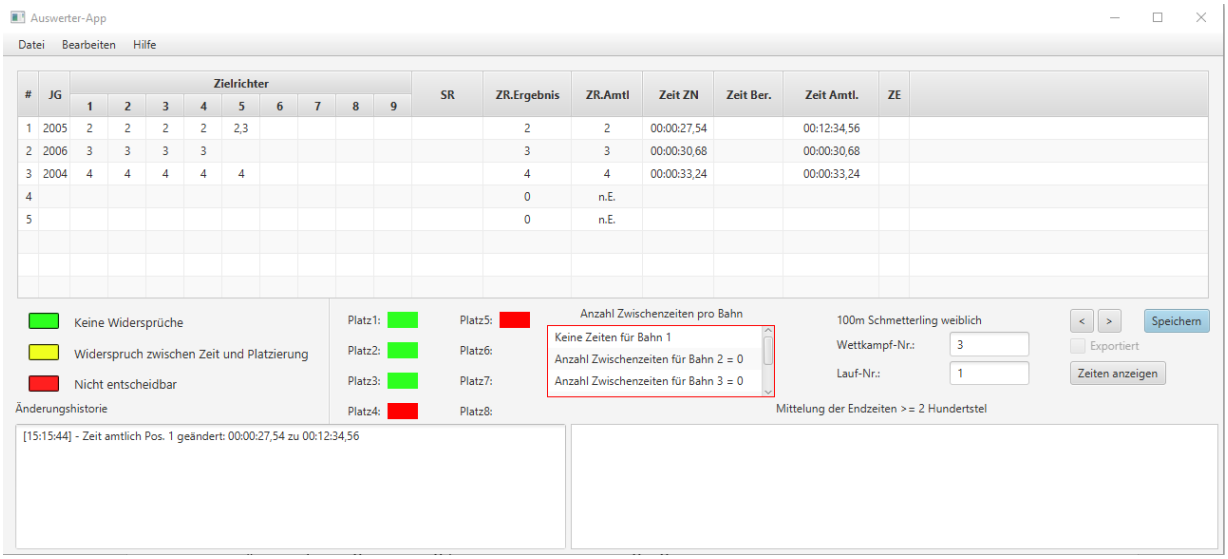


Abbildung 61: Ansicht Auswerter-App

Datei: Die Option „Wettkampf öffnen…“ sollte ursprünglich eine Funktion haben. Diese wurde aber, aufgrund einer Designentscheidung, verworfen. Da der Kontrollrechner alle Daten schickt und dem Auswerter im Vorfeld nicht alle Wettkämpfe bekannt sein müssen, ist diese Funktion gestrichen worden. „Beenden“ beendet das Programm.

Bearbeiten: Die Adresse des Kontrollrechners lässt sich hier auch ändern, während das Programm läuft, indem man auf Einstellungen klickt. Falls die Verbindung während des Betriebs abbricht, kann ein Reconnect mit einem Klick auf Einstellungen > Übernehmen probiert werden. Falls eine Adresse oder ein Port angegeben wird, zu der keine Verbindung aufgebaut werden kann, werden die Eingaben nicht übernommen.

Hilfe: (Noch) keine Funktion.

In der Tabelle werden alle Daten angezeigt, welche vom Kontrollrechner empfangen werden.

Bezeichnung	Beschreibung
#	Platzierungen 1 bis 5.
JG	Jahrgang des Schwimmers auf der Bahn.
Zielrichter	Die Abstimmung der Zielrichter.
SR	Schiedsrichterfeld
ZR.Ergebnis	Zielrichter Ergebnis. Hier trägt der Algorithmus automatisch sein berechnetes Ergebnis ein.
ZR.Amtlich	Zielrichter Amtlich. Hier wird das Ergebnis aus ZR.Ergebnis übernommen, kann aber manuell überschrieben werden.
Zeit ZN	Zeit Zeitnehmer, zeigt die Endzeit an, die der Zeitnehmer genommen hat. Die Zeit gehört zu der unter ZR.Amtlich eingestellten Bahn.

Zeit Ber.	Zeit berechnet, zeigt falls nötig gemittelte Zeiten an. Zeiten werden gemittelt, wenn Platzierung und Zeit im Widerspruch zueinanderstehen aber dennoch die Platzierung nicht geändert wird.
Zeit Amtl.	Amtliche Zeit, wird automatisch mit dem Inhalt von Zeit ZN gefüllt, es sei denn es gibt einen Mittelwert. In dem Fall wird der Mittelwert genommen.
ZE	Zielrichterentscheid; Hier sollte ursprünglich etwas stehen, falls die Platzierung ein Zielrichterentscheid war. Dieses Feature wurde jedoch nicht mehr implementiert.

Tabelle 2 Übersicht Daten, die vom Kontrollrechner empfangen werden

Die Spalten SR (Schiedsrichter), ZR.Amtl. und Zeit Amtl. sind bearbeitbar. In SR kann eingetragen werden, was der Schiedsrichter gesehen hat, falls ein Ergebnis unklar ist. Dieses Feld hat keinerlei Einfluss auf die automatische Ermittlung des Ergebnisses. ZR.Amtl. erwartet eine Zahl von 0 bis 8. Eine 0 setzt diese Platzierung auf „n.E." und entfernt sie effektiv aus der Auswertung. Zeit Amtl. erwartet die Eingabe einer Zeit.

Unter der Tabelle befindet sich auf der linken Seite eine Legende. Rechts davon ist für jede Platzierung eine farbliche Markierung. Im Feld „Anzahl Zwischenzeiten pro Bahn" ist auf einen Blick sichtbar, wie viele Zeiten jede Bahn hat und ob eine fehlt. Haben die Bahnen unterschiedliche Anzahlen wird der Rand des Feldes rot markiert. Auf der rechten Seite befinden sich Buttons und Felder, um zwischen den Wettkämpfen vor und zurück zu navigieren. Mithilfe der Felder lässt sich direkt zu einem Wettkampf springen. Mit einem Klick auf „Zeiten anzeigen" können alle Zwischenzeiten eingesehen werden. Im Zwischenzeiten-Fenster können angewählte Zwischenzeiten mit einem Klick auf „Löschen" gelöscht werden, falls diese irrtümlich aufgenommen wurde. Ein Klick auf Speichern exportiert den derzeitigen Wettkampf in das Lenex Format, welches dann in EasyWK importiert werden kann.

Am untersten Rand des Fensters befindet sich links eine Änderungshistorie, welche lediglich alle manuell gemachten Änderungen erfasst. Rechts davon befindet sich eine Übersicht, welche Platzierungen im Falle einer Zeitmittelung auflistet, bei denen die Ausgangszeiten um mehr als zwei hundertstel Sekunden auseinander liegen.

A.5 Lenex (FileChangeListener)

Um den Lenex/EasyWK-Austausch zu ermöglichen muss der FileChangeListener verwendet werden. Das Projekt inkl. Sourcecode und die bereits erstellte FileChangeListener.jar-Datei befinden sich auf der Gitlab-Website https://gitlab.repo.digitech.hs-emden-leer.de/rkuche/filechangelistener.git.

Zuerst muss in EasyWk ein ALGE-Austauschordner selektiert werden. Dies erfolgt in EasyWk über den Menüpunkt Protokoll → Ergebnisse → Datenaustausch → Anbindung ALGE-Timing. Es ist zu empfehlen einen neuen Ordner zu erstellen und diesen direkt auszuwählen. Dort wird von EasyWk automatisch die Dateien splash_receive.txt und splash_send.txt erzeugt. Die von der Auswerter-App generierten Lenex-Austauschdateien müssen sich beim Import in dem vorher ausgewählten Ordner befinden. Ebenfalls die FileChangeListener.jar-Datei, welcher als Controller zwischen den Austauschdateien und Lenex-Austauschdateien fungiert wird in dem Ordner platziert.

Um den FileChangeListener ausführen zu können muss vorerst ein Terminal (Konsole) gestartet werden. Dort erstmal wird mittels dem `cd`-Befehl in das Austauschverzeichnis navigiert, in welchem sich auch der FileChangeListener befindet. Vor dem Import ist es nötig diesen mit folgendem Kommando zu starten: `java -jar FileChangeListener.jar start`

In EasyWk kann nun mittels des Buttons „Zeiten holen" die derzeitige Veranstaltung geladen werden. Auf der Konsole des FileChangeListener können Informationen zum Import abgelesen werden, sprich

ob die zu importierende Datei gefunden werden konnte, ob ein Onlinestatus bei EasyWk übermittelt wurde usw.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellenverzeichnis

[1] HiveMQ Team (2014): *MQTT 101 – How to Get Started with the lightweight IoT Protocol*. In: HiveMQ. [https://www.hivemq.com/blog/how-to-get-started-with-mqtt/, 10.09.2018]

[2] Online-Shop sparkfun: *Raspberry Pi Zero W*. In: sparkfun. [https://www.sparkfun.com/products/14277, 08.02.2019]

[3] Online-Shop ThePiHut: *Raspberry Pi 3 Model B+*. In: ThePiHut. [https://thepihut.com/products/raspberry-pi-3-model-b-plus, 08.02.2019]

[4] Abels, Henning; Boucif, Noureddine; Chen, Xingjian; Eikens, Oliver; Eilts, Joachim; Golchert, Frederik; Mass, Marcel; Meyer, Christoph; Nannen, Christian; Popp, Sebastian; Seckler, Gerrit; Siemer, Alexander (2018): *Projektdokumentation: Entwurf und Implementierung eines Systems zur Erfassung, Bearbeitung und Auswertung von Wettkampfdaten für Schwimmwettkämpfe*. Hochschule Emden/Leer

[5] Mahmood, Aneeq; Exel, Reinhard; Trsek, Henning; Sauter, Thilo (2017): *Clock Synchronization Over IEEE 802.11 - A Survey of Methodologies and Protocols*. In: IEEE Transactions on Industrial Informatics, Vol 13, No. 2. S. 915-916

[6] Delecroix, Nicolas (2017): *IEEE 1588 PTP and Analytics on the Cisco Nexus 3458 Switch*. In: Cisco. [https://www.cisco.com/c/en/us/products/collateral/switches/nexus-3000-series-switches/white-paper-c11-731501.html, 06.02.2019]

[7] H2 Database Engine: http://www.h2database.com/html/main.html

[8] EasyWk - Software für den Schwimmsport: https://www.easywk.de

[9] Spring Ecosystem: https://spring.io

[10] MySQL Workbench: https://www.mysql.com/de/products/workbench/

[11] Offizielle Lenex Spezifikationen (englisch): https://wiki.swimrankings.net/images/6/62/Lenex_3.0_Technical_Documentation.pdf

[12] http://fxexperience.com/controlsfx/features/

[13] https://github.com/joffrey-bion/fx-gson

[14] https://www.balena.io/etcher/

[15]https://sourceforge.net/projects/win32diskimager/

[16] https://www.putty.org/

[17] https://github.com/eclipse/paho.mqtt.c

[18] https://sourceforge.net/projects/win32diskimager/

[19] Wettkampfbestimmungen des Deutschen Schwimmverbandes - Fachteil Schwimmen (WB-FT SW) vom 14.12.2018 http://www.dsv.de/fileadmin/dsv/documents/schwimmen/Amtliches/Wettkampfbestimmungen-Fachteil_Schwimmen_2018_12_14.pdf