4. Regionalwettbewerb „Jugend forscht“ in Jena, Fachgebiet Mathematik/Informatik

Kanu s.a.M. - Wettkampferfassung und -auswertung beim Kanusport mittels eines kostenlos und einfach einrichtbaren Softwaresystems

Eric Ackermann
Carl-Zeiss-Gymnasium Jena

15. Januar 2017

betreut durch:
Herrn Bernd Schade

Kurzfassung

Professionelle Zeitmesssysteme für den Kanusport sind teuer und meist kompliziert einzurichten. Kleinere Vereine, die sich ein solches System nicht leisten können, sind auf Papierlisten für Strafzeiten und die Zeitmessung per Hand angewiesen. Dies bedeutet einen hohen Zeit- und Organisationsaufwand bei Training und Wettbewerben.
Ziel dieses Projekts ist die Entwicklung eines Softwaresystems, welches auf vorhandener bzw. kostengünstig anzuschaffender Hardware funktioniert, einfach zu bedienen ist und die Zeitmessung für Kanu-Sportvereine optimiert. Im Mittelpunkt steht die Entwicklung einer Android-App, die das Eintragen von Strafzeiten sowie Start- und Zielzeit für die einzelnen Startnummern und Tore mit einem Klick leistet. Die in der App erhobenen Daten werden über ein lokales WLAN-Netzwerk unter Zuhilfenahme eines lokalen PHP- und MySQL- kompatiblen Servers mit einem zentralen Laptop synchronisiert. Auf diesem läuft eine eigens entwickelte Desktopanwendung, die alle Daten zusammenführt, jederzeit anzeigt und nach dem Wettkampf mit einem Klick übersichtlich auswertet.

Abkürzungsverzeichnis

s.a.M. semi-automatisches Messsystem

PHP Hypertext PreProcessor

SQL Structured Query Language

IP Internet-Protokoll

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

POI Poor Obfuscation Implementation

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Zeitmessung im Kanusport/Problemstellung . . . 1

1.2 Konzeption . . . 2

2 Vorgehensweise **3**

2.1 Grundprinzip der Vernetzung der verschiedenen Geräte . . . 3

2.2 Aufbau und Funktionsweise der verwendeten MySQL-Datenbank . . . 4

2.3 Das Hauptprogramm „Kanu s.a.M.“ . . . 6

2.3.1 Features . . . 6

2.3.2 Anlegen und Verwalten der Datenbank . . . 7

2.3.3 Datenverarbeitung und Auswertung . . . 7

2.4 Die Android-App „Kanu s.a.M. App“ . . . 8

2.4.1 Funktionen . . . 8

2.4.2 Umsetzung des Verbindungsaufbaus . . . 8

2.5 Der PHP-Server . . . 9

2.5.1 Aufgabe . . . 9

2.5.2 Umsetzung des Zugriffes auf die Datenbank . . . 10

2.6 Technische Umsetzung der Synchronisation . . . 10

2.7 Beschreibung des Ablaufs einer Verwendung des Softwaresystems . . . 11

2.8 Aufgetretene Probleme und ihre Lösungen . . . 12

3 Zusammenfassung und Schluss **14**

3.1 Umsetzung der Konzeption . . . 14

3.2 Ausblick . . . 15

Glossar **16**

Literatur **18**

4 Anhang **20**

4.1 Screenshots . . . 20

4.1.1 Hauptprogramm . . . 20

4.1.2 App . . . 23

4.1.3 Protokolle . . . 25

4.2 Sourcecode . . . 25

Abbildungsverzeichnis

1.1 Sportler beim Befahren eines Messtores[1] . . . 1
1.2 Fotofinish und Laufzeitmessung beim Zeitmesssystem „Professional“ von IMAS, vgl. [4] . 2
2.1 Schema der Vernetzung der verschiedenen Geräte, vgl. [3, 12, 9, 2, (Illustrationen)] . . . 3
2.2 Übersicht über die Datenbank „**android_connect**“ und ihre Tabellen in einer Beispielbelegung, Darstellung durch verwendetes Datenbankverwaltungsprogramm PHPMyAdmin (Teil von XAMPP) . . . 4
2.3 Beispielbelegung der Datenbanktabelle „**allgemein**“ . . . 5
2.4 Beispielbelegung der Datenbanktabelle „**namen**“ . . . 5
2.5 Beispielbelegung der Tabelle eines Messtores . . . 5
2.6 Beispielbelegung der Tabelle für die Auswertung . . . 5
2.7 Ausschnitt aus einer Auswertungsdatei mit Beispieldaten, dargestellt von LibreOffice . . . 7
4.1 Konfigurationsfenster des Hauptprogramms mit Beispieleingaben . . . 20
4.2 Wahl der Kategorie, welche starten soll . . . 20
4.3 Zuordnung der Messtore zu Messstation 1 . . . 20
4.4 Hauptfenster vor Start des ersten Starters . . . 21
4.5 Hauptfenster während eines Laufs . . . 21
4.6 Hauptfenster bei Laufende . . . 22
4.7 Auswahl der Starter, die zum Wiederholungslauf antreten . . . 22
4.8 App vor dem Verbindungsaufbau . . . 23
4.9 App bei der Funktionswahl . . . 24
4.10 App im Startmodus bei Auswahl der Startnummer (Stoppmodus ähnlich) . . . 24
4.11 App beim Eintragen einer Strafzeit . . . 25
4.12 Protokoll eines Programmlaufs mit Beispielausgaben . . . 25

Tabellenverzeichnis

2.1 Features des Hauptprogramms „Android_Connector“ 6
2.2 Features der App „Android Connector App HTTP“ 8

1 Einleitung

1.1 Zeitmessung im Kanusport/Problemstellung

Beim Kanusport müssen sowohl im Training als auch im Wettkampf viele Zeiten gemessen und zugeordnet werden. Dies erfolgt bei kleinen Vereinen, so auch beim Jenaer Kanu- und Ruderverein und beim Kanu-Slalom-Teil des SV Schott, bisher meist manuell. Beispielsweise werden an der Jenaer Saale auf einer Strecke von 200 m insgesamt 18-25 Messtore (siehe Abbildung 1.1) festgesetzt. Diesen sind 8 Messstationen zugeteilt. Von denen aus können Helfer mehrere Tore überblicken und Startern Strafzeiten für falsches Passieren oder Berühren der Torstäbe zuweisen, wie dies laut den Wettkampfregeln vorgeschrieben ist. Die erhobenen Strafzeiten müssen bei bis zu 27 einzelnen Rennen in jeweils zwei Läufen für viele Sportler erfasst und zugeordnet werden.

Abbildung 1.1: Sportler beim Befahren eines Messtores[1]

Zur Wettkampfauswertung werden sie zu den gemessenen Laufzeiten addiert. Üblicherweise zählt der beste Lauf. Die Gesamtzeiten aller Starter einer Kategorie (festgelegt nach Alter und Bootstyp) werden verglichen, um eine Rangfolge zu ermitteln.
All dies bedeutet einen hohen Organisationsaufwand in der Wettbewerbsauswertung. Bisher wurden alle Zeiten per Hand aufgeschrieben und z.B. in Excel ausgewertet. Dieses Verfahren erhöht den Zeitaufwand, ist fehleranfällig und langwierig.
Es existieren kommerzielle Systeme zur automatischen Zeiterfassung. Eine vollwertige Anlage ist z.B. mit dem „Time System Professional“ von IMAS[5, 6] erwerbbar. Die Zeiterfassung erfolgt per Kamera (im Ziel, siehe Abbildung 1.2) und optional einer automatischer Startanlage bzw. einem Startkoffer. Es gibt eine Funktion zur Erfassung von Zwischenzeiten an den Messstationen. Dafür muss an jeder solchen Station ein Helfer einen Laptop bedienen. Ein anderes Programm („Regatta Manager“) verwaltet die Starter. Die aktuellen Laufzeiten können ins Internet gestreamt oder lokal auf LCD-Anzeigen ausgegeben werden. Die Laptops, Kameras und anderen Anlagen müssen über LAN oder WLAN vernetzt werden.

Abbildung 1.2: Fotofinish und Laufzeitmessung beim Zeitmesssystem „Professional“ von IMAS, vgl. [4]

Das System kann jedoch keine Strafzeiten synchronisieren, diese werden überhaupt nicht erfasst.
IMAS gibt keinen Festpreis an, da Teile des Systems wie Startanlage und Kamera optional gegen Mehrkosten erworben werden müssen. Ein vergleichbares System von Lynx kostet mindestens 41.595,- € zzgl. MwSt, in der günstigsten Variante noch 11.995,- € zzgl. MwSt[10]. Dieses kann ebenfalls keine Strafzeiten synchronisieren.
Es wurde bei einer gezielten Recherche kein kostengünstigeres System für kleine Vereine gefunden. Kommerzielle Systeme bieten auf der einen Seite meist mehr Funktionen als grundlegend nötig und sind deswegen teuer. Auf der anderen Seite lassen sie wichtige Features, besonders die Erfassung von Strafzeiten, vermissen. Der Grund dafür ist, dass sie für eine andere Form des Kanusports gedacht sind (Kanu-Rennsport). Bei dieser starten alle Sportler parallel und fahren eine gerade Strecke möglichst schnell entlang. Für die Sportart Kanu-Slalom geeignete Systeme waren selbst bei langer Recherche auf dem freien Markt nicht zu finden und sind meist Spezialanfertigungen gegen noch höheren Preis, welche auf einem der o.g. Systeme basieren.
Die dieser Arbeit zugrunde liegende Frage lautet deshalb: Lässt sich ein Zeitmesssystem für den Kanuslalom entwickeln, das mit freier oder selbst entwickelter Software und günstiger Hardware funktioniert?

1.2 Konzeption

Das zu entwickelnde Softwaresytem soll das Problem der Starterverwaltung, Wettkampfzeiterhebung, Strafzeitensynchronisation und Wettkampfauswertung beim Kanuslalom möglichst günstig und einfach lösen. Dafür werden als Hardware nur Android-Smartphones und ein einziger Windows-Rechner verwendet. Smartphones sind weit verbreitet, und auch mindestens ein Windows-Laptop ist in jedem Kanuverein verfügbar. Auf diesen zu installierende oder zu verwendende Software soll gratis verfügbar sein. Die Geräte werden ggf. mit einem drahtlosen Netzwerk untereinander verbunden. Aus Gründen der Manipulierbarkeit muss das System ohne Internetverbindung funktionieren.

2 Vorgehensweise

2.1 Grundprinzip der Vernetzung der verschiedenen Geräte

Der zentrale Laptop ist das wichtigste Gerät im entworfenen Softwaresystem „Kanu s.a.M.“ (für semi-automatisches Messsystem). Es handelte sich dabei bei allen Tests um ein Windows- Gerät. Nötige Software ist von Windows XP bis Windows 10 lauffähig und im Rahmen der Grundfunktionalität auch zu Apple- oder Linuxgeräten kompatibel. Einige Zusatzfunktionen gehen dann jedoch verloren.
Der Laptop muss ein WLAN-Netz für die Vernetzung der Geräte aufbauen. Ein Internetzugriff ist jedoch nicht nötig.

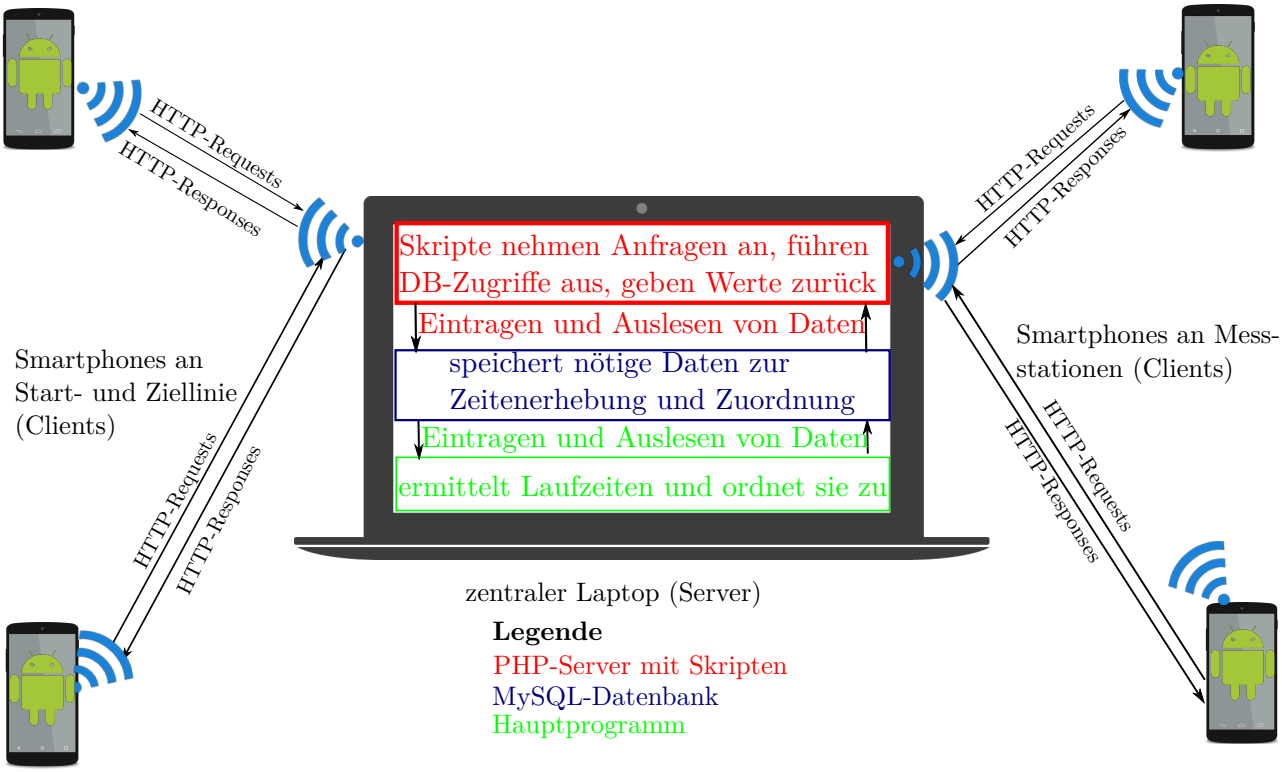


Abbildung 2.1: Schema der Vernetzung der verschiedenen Geräte, vgl. [3, 12, 9, 2, (Illustrationen)]

Die wichtigste Fremdsoftware ist eine MySQL-Datenbank, die alle Daten speichert, welche für die Verbindung der einzelnen Geräte gebraucht werden (siehe Abbildung 2.1).
Diese Datenbank wird durch MariaDB verwaltet (ein freier Ableger von MySQL mit gleicher technischer Grundlage). Parallel zur Datenbank läuft ein PHP-Server, hier der Apache Webserver.
Die Helfer halten sich mit ihren Smartphones in ihren Messstationen auf. Auf den Smartphones wird eine eigens entwickelte Android-App ausgeführt, die das Starten und Stoppen der Laufzeitmessung oder das Eintragen von Strafzeiten ermöglicht. Die App kommuniziert über den PHP-Server mit der Datenbank.
Auf dem Hauptrechner läuft das in JavaFX geschriebene Hauptprogramm „Kanu s.a.M.“. Es verbindet sich direkt mit der Datenbank und kann seine Laufzeit- und Strafzeitdaten über sie mit der App synchronisieren. Es verwaltet zudem Starter und Rennen, ermittelt während eines Laufs immer die aktuellen Laufzeiten der laufenden Starter und wertet am Ende den Wettkampf aus.

2.2 Aufbau und Funktionsweise der verwendeten MySQL-Datenbank

MariaDB kann nur auf einem Webserver ausgeführt werden. Der zentrale Laptop, der hier auch als Server fungiert, wird darum zum Anlegen eines solchen genutzt. Dafür ist die Installation der Software XAMPP erforderlich. Sie simuliert einen Server, der unter Anderem das Verwalten einer Datenbank und die Ausführung von PHP ermöglicht. Der Webserver bleibt dabei rein lokal. Er kann innerhalb des lokalen Netzes adressiert werden, aber nicht vom Internet aus, solange kein Gerät eine Internet-Verbindung besitzt. Dies dient der Datensicherheit und dem Schutz vor Manipulationen. Eine direkte Internet-Zugriffsmöglichkeit ist nicht zu empfehlen, da XAMPP für den einfachen und schnellen lokalen Betrieb konzipiert ist und nicht für den Betrieb als echter Server. Viele von ihm vorgenommene Konfigurationsdetails der Teilprogramme dienen mehr der Geschwindigkeit als der Sicherheit.
Die Datenbank (siehe Abbildung 2.2) enthält die Tabellen „**namen**“, „**allgemein**“, für jede Messstation eine, sowie eine für die Auswertung („**auswertung**“). Die Startnummer ist in allen Tabellen Primärschlüssel.

Abbildung 2.2: Übersicht über die Datenbank „**android_connect**“ und ihre Tabellen in einer Beispielbelegung, Darstellung durch verwendetes Datenbankverwaltungsprogramm PHPMyAdmin (Teil von XAMPP)

Die Tabelle „**allgemein**“ speichert Informationen, die für die Initialisierung und korrekte Eintragungen der Handy-App relevant sind. Namentlich (siehe Abbildung 2.3) sind dies:

1. ob der erste Starter bereits gestartet ist (ab dann Eintragung von Strafen möglich in der App)
2. die Nummer des aktuellen Laufs (wird zur Orientierung in der App eingeblendet)
3. die Tore, welche zu den einzelnen Messstationen gehören
4. die Startzeit des Wettkampfes als Java-Timestamp und
5. die Anzahl der Messstationen, damit Benutzer der App sich an jede davon anmelden können.

	Attribut	Wert
Bearbeiten Kopieren Löschen	gestartet	true
Bearbeiten Kopieren Löschen	lauf	0
Bearbeiten Kopieren Löschen	Messstation_0_Tore	0|3|5|7
Bearbeiten Kopieren Löschen	Messstation_1_Tore	2|4
Bearbeiten Kopieren Löschen	Messstation_2_Tore	1|6
Bearbeiten Kopieren Löschen	Startzeit	1482847016286
Bearbeiten Kopieren Löschen	Zahl_Stationen	3

Abbildung 2.3: Beispielbelegung der Datenbanktabelle „**allgemein**“

Die Tabelle „**namen**“ (siehe Abbildung 2.4) speichert Namen und Startnummern der Starter für die Anzeige oder Auswahl der Starter in der App, eine ggf. von der App per Script eingetragene Start- oder Zielzeit als PHP-Timestamp sowie eine Zusammenfassung der einzelnen Läufe für eine spätere Auswertung als String. Zudem ist die Information, welchen Lauf ein Wiederholungslauf ggf. ersetzt, enthalten.

	Namen	Startnummer	Startzeit	Zielzeit
Bearbeiten Kopieren Löschen	Hans-Dietrich Genscher	10	1482847027.7587	1482847166.0294
Bearbeiten Kopieren Löschen	Franz Franz	27	1482847020.1146	1482847159.4347

Lauf_1	Lauf_2	Lauf_Wiederholung	Wiederholung_ersetzt
2:18,193|104|4:2,193Tor: 1|Strafe: 50|Tor: 2|Straf...	*NULL*	*NULL*	*NULL*
2:19,320|104|4:3,320Tor: 1|Strafe: 0|Tor: 2|Strafe...	*NULL*	*NULL*	*NULL*

Abbildung 2.4: Beispielbelegung der Datenbanktabelle „**namen**“

Die Tabellen für die jeweiligen Messtore (Nummer in Computerzählweise ab 0, siehe Abbildung 2.5) speichern an diesem Tor je die Startnummer und ihre Strafzeit in Sekunden.

	Startnummer	Zeitpunkt
Bearbeiten Kopieren Löschen	10	50
Bearbeiten Kopieren Löschen	27	0

Abbildung 2.5: Beispielbelegung der Tabelle eines Messtores

Die Tabelle „**auswertung**“ ist angelegt wie die Tabelle „**namen**“, nur dass sie alle Startnummern enthält und bei allen die Kategorie mit angegeben ist. Es gibt dort keine Spalten für Start- und Zielzeit.

Namen	Startnummer	Kategorie	Lauf_1	Lauf_2	Lauf_Wiederholung	Wiederholung_ersetzt
Max Mustermann	1	Elite	2:50,901|52|3:42,901|Tor: 1|Strafe: 0|Tor: 2|Straf...	2:50,222|6|2:56,222|Tor: 1|Strafe: 2|Tor: 2|Strafe...	*NULL*	*NULL*
Karl Schmidt	15	Pro			*NULL*	*NULL*
Franz Franz	27	Ober-Elite			*NULL*	*NULL*
Franz Schuster	3	Elite	2:51,237|4|2:55,237|Tor: 1|Strafe: 2|Tor: 2|Strafe...	2:56,114|6|3:2,114|Tor: 1|Strafe: 2|Tor: 2|Strafe:...	*NULL*	*NULL*
Herrmann Mann	35	Pro			*NULL*	*NULL*

Abbildung 2.6: Beispielbelegung der Tabelle für die Auswertung

2.3 Das Hauptprogramm „Kanu s.a.M.“

Das auf dem zentralen Laptop laufende Programm wurde entwickelt mit NetBeans IDE 8.2 und läuft in einer Java Virtual Machine. Damit ist es zu allen Betriebssystemen kompatibel, die von Java unterstützt werden (Windows, Linux, MacOS, Solaris u.ä.). Es funktioniert „semi-automatisch“, muss also vor dem Wettkampfstart mit den Starterdaten, der Torkonfiguration und den Netzwerkparametern eingerichtet werden und benötigt nach dem Start nur noch einen Bediener, der den nächsten Lauf startet oder die nächste Kategorie auswählt. Es ist über die Datenbank, den PHP-Server und WLAN indirekt mit der App verbunden.

2.3.1 Features

Tabelle 2.1 zeigt alle Features des Hauptprogramms zusammengefasst. Bei Interesse an der Umsetzung steht der Quellcode gern zum Nachlesen zur Verfügung. Klassen und Methoden, welche die Features umsetzen, sind angegeben.

Feature	**Realisierung mittels Java-Klasse(n) und Methode(n)**
Einlesen von Starterdaten aus Excel-Dateien	Prozedur ConfigWindowController.readExcel(), Klasse ExcelReader, Library Apache POI
Anzeige, Veränderbarkeit eingelesener Daten	Klasse ConfigWindowController, Klasse javafx.scene.control.TableView (und assoziierte Klassen), Klasse Person, Klasse EditabletableCell
automatisches Leeren der Datenbank	Prozedur MySQLConnection.reset
automatisches Starten und Stoppen von XAMPP	Prozeduren ConfigWindowController.xampp_start, xampp_stopp
automatisches Kopieren von nötigen PHP-Scripten, Überschreiben der PHP-Konfiguration	Prozedur ConfigWindowController.overridePHPConfig
Erstellen von Sicherungsprotokollen	Prozedur MySQLConnection.werteFesthalten
Prüfen des Netzwerkzugriffs, Ermittlung der eigenen IP-Adresse	Klasse NetworkUtil
Zugriff auf MySQL-Datenbank (allgemein)	Klasse MySQLConnection, Klasse com.mysql.jdbc.Driver (Teil der verwendeten Library), Klasse java.sql.Statement (für eigentliche Abfragen)
Verwaltung der Starter, Zuordnung zu Kategorien, Start dieser	Klasse ConfigWindowController
Verwaltung der Daten der einzelnen Läufe, Start und Stopp dieser	Klasse FXMLDocumentController
Anzeige der aktuellen Laufaten der jeweiligen Starter	Klasse MySQLConnection, Klasse javafx.scene.control.ListView
Anzeige der Daten im angegebenen Intervall	Klasse java.util.concurrent.ScheduledExecutorService, Klasse java.util.concurrent.ScheduledFuture, Klasse java.lang.Runnable
Auswertung der Rennen	Prozedur ConfigWindowController.printResults, Funktion MySQLConnection.getAuswertung

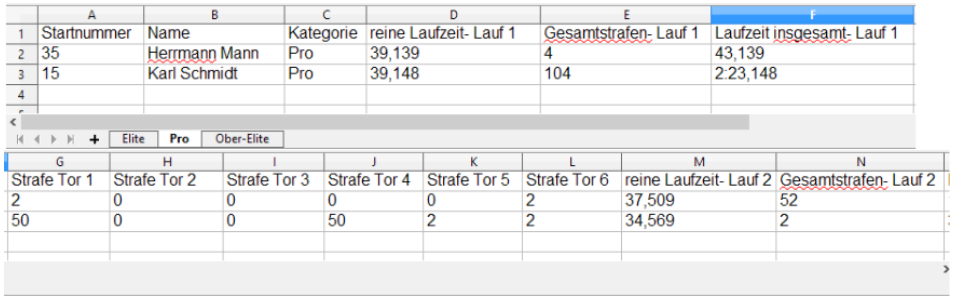
Tabelle 2.1: Features des Hauptprogramms „Android_Connector“

2.3.2 Anlegen und Verwalten der Datenbank

Der Verbindungsaufbau zur Datenbank erfolgt im Konstruktor der Klasse **MySQLConnection**. Dieser speichert übergebene Konfigurationswerte global in der aktuellen Instanz der Klasse und legt eine Instanz der Klasse **java.sql.Connection** namens **conn** an. Diese stellt die eigentliche Verbindung zur Datenbank her. Ab jetzt können Abfragen ausgeführt werden. Es gibt dabei 2 Arten von Datenbankbfragen in SQL: Schreibafragen, die z.B. über den Befehl **CREATE** eine neue Tabelle anlegen oder über **UPDATE** Werte in einer Tabelle ändern, und Leseabfragen, die ein Ergebnis liefern sollen.
conn stellt nun die Methode **createStatement** bereit, welche eine Instanz von **java.sql.Statement** zurückgibt. **Statement** stellt die Methode **execute** bereit, welche einen übergebenen String als Datenbankabfrage in der adressierten Datenbank ausführt. Sie lässt nur Schreibzugriffe zu. Für Leseabfragen existiert die Methode executeQuery, welche eine übergebene SQL-Abfrage in der Datenbank ausführt und eine Instanz von **java.sql.ResultSet** zurückgibt. Dieses repräsentiert die von der Datenbank erzeugte Antworttabelle und kann zeilenweise gelesen werden. Der Zugriff auf einzelne Spalten der Zeile, welche eine gewünschte Information enthalten, und eine Speicherung als String für eine weitere Verarbeitung ist ebenfalls möglich. Andere Datenformate wie **Boolean** oder **Timestamp** (in SQL-Syntax) müssen erst geparst oder umgewandelt werden, da sie von MySQL in anderen Formaten gespeichert werden.

2.3.3 Datenverarbeitung und Auswertung

Das Ende des Wettkampfs wird dadurch erkannt, dass keine Kategorien verbleiben, die noch nicht gestartet sind. Alle Läufe müssen dann abgeschlossen sein.
Zunächst wird die Prozedur **printResults** im **ConfigWindowController** aufgerufen. Diese ruft nun zuerst die Methode **getAuswertung** von **MySQLConnection** auf. Sie liest die gespeicherten Laufzusammenfassungen aus der Tabelle „**auswertung**“ der Datenbank aus, welche vorher aus „**namen**“ kopiert wurden (siehe Abbildung 2.6). Grund für das Kopieren ist, dass alle Tabellen außer „**auswertung**“ bei jedem neuen Lauf gelöscht werden. Die Prozedur ersetzt Läufe ggf. bereits durch den Wiederholungslauf (wie in der Datenbank angegeben) und gibt ein Array mit allen Werten aller Starter zurück.
printResults nimmt nun die Laufzusammenfassungen mittels einer Untermethode auseinander und extrahiert die reine Laufzeit, die Gesamtstrafen, die Gesamtlaufzeit und die Strafen an den einzelnen Toren. Die Library **Apache POI** wird jetzt verwendet, um aus den Werten eine Excel-Datei (siehe Abbildung 2.7) zu erzeugen. Dabei wird für jede Kategorie eine Mappe angelegt, welche die Angaben für die jeweiligen Starter dieser enthält. Eine Auswertung ist nun z.B. per Funktion „sortieren“ möglich.



	A	B	C	D	E	F
1	Startnummer	Name	Kategorie	reine Laufzeit- Lauf 1	Gesamtstrafen- Lauf 1	Laufzeit insgesamt- Lauf 1
2	35	Herrmann Mann	Pro	39,139	4	43,139
3	15	Karl Schmidt	Pro	39,148	104	2:23,148
4						

Elite | Pro | Ober-Elite

G	H	I	J	K	L	M	N
Strafe Tor 1	Strafe Tor 2	Strafe Tor 3	Strafe Tor 4	Strafe Tor 5	Strafe Tor 6	reine Laufzeit- Lauf 2	Gesamtstrafen- Lauf 2
2	0	0	0	0	2	37,509	52
50	0	0	50	2	2	34,569	2

Abbildung 2.7: Ausschnitt aus einer Auswertungsdatei mit Beispieldaten, dargestellt von LibreOffice

2.4 Die Android-App „Kanu s.a.M. App“

Es handelt sich bei der App, die auf den Smartphones der Helfer ausgeführt wird, um eine reine Android-App. Sie wurde entwickelt mit Android Studio 2.2, der offiziellen IDE von Google.
Die verwendete Programmiersprache ist auch hier Java. Die Interpretation auf den Smartphones erfolgt durch Dalvik Virtual Machine.
Sie ist lauffähig ab Android 2.3.3 auf allen Smartphones und Tablets mit WLAN. Auch Android Wear-Unterstützung ist theoretisch möglich, sodass die App auch auf Smartwatches oder einer Google Glass ausgeführt werden könnte.

2.4.1 Funktionen

Tabelle 2.2 zeigt alle Features der App zusammengefasst. Bei Interesse an der Umsetzung steht der Quellcode gern zur Verfügung. Klassen und Methoden sowie PHP-Scripte sind den durch sie realisierten Features zugeordnet.

Feature	Realisierung mittels
Verbindungsaufbau zur Datenbank über den PHP-Server	Activity ConnectionActivity, Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Abfrage_Stationen.php`
Anzeige des aktuellen Laufs	Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Abfrage_Lauf.php`
Handstarten eines ausgewählten Starters	Activity StartActivity, Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Abfrage_Startnummern.php`, PHP-Script `Startzeit_eintragen.php`
Handstoppen eines ausgewählten Starters	Activity StoppActivity, Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Abfrage_Startnummern.php`, PHP-Script `Zielzeit_eintragen.php`
Anzeigen der vergangenen Zeit seit Wettbewerbsstart	Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Abfrage_Startzeit.php`, Klasse Thread, Instanz „uhr“
Auswahl der aktuellen Startnummer und ihren Strafen an den lokalen Toren	Activity MainActivity, Klasse android.widget.Spinner, Klasse android.widget.TableLayout
Eintragen der gewählten Strafzeiten an den gewählten Toren für die gewählte Startnummer	Methode MainActivity.onClick, Klasse HTTP_Connection, Interface AsyncResponse, PHP-Script `Zeit_eintragen.php`

Tabelle 2.2: Features der App „Android Connector App HTTP“

2.4.2 Umsetzung des Verbindungsaufbaus

Android hat „von Haus aus“ keine Möglichkeit, direkt auf MySQL-Datenbanken auf fremden Geräten zuzugreifen. Ein solcher Verbindungsaufbau könnte auch von einer Firewall geblockt werden. Android kann aber HTTP-GET-Anfragen senden.
Das Problem dabei: solche Anfragen dürfen ab Android 3.0 nicht mehr auf dem Main- oder UI-Thread ausgeführt werden, siehe hierzu [8].

Der Hintergrund ist: längere Operationen wie Netzwerkzugriffe (mit Verzögerungen der Antwort durch Timeouts bis mehrere Sekunden) würden den Thread blockieren und somit die GUI „einfrieren“. Dies stellt eine unangenehme Situation für den Benutzer dar, der denkt, die App hätte sich „aufgehängt“. Außerdem zeigt Android nach 5 Sekunden „Frost“ eine App-not-responding-Meldung an und bietet an, die App zu beenden.
Eine Alternative hierzu bietet das Interface **AsyncTask**, das zunächst beerbt werden muss (so geschehen in Klasse **HTTPConnection**). Es legt einen zweiten Thread an, der parallel zum Mainthread läuft und auf dem Abfragen ausgeführt werden können. Damit blockiert er die UI nicht. **AsyncTask** stellt die abstrakte Methode **doInBackground** bereit, welche auf diesem zweiten Thread läuft.
Die Klasse **URLConnection** kann unter Verwendung von **URL*openConnection()** nun die als String übergebene HTTP-Abfrage senden. Es handelt sich dabei um eine GET-Anfrage an ein Script des PHP-Servers auf dem zentralen Laptop, zusammengesetzt nach dem Schema `http://IP-Adresse_des_Rechners/AndroidConnectorAppHTTPScripts/Scriptname.php?Parameter="Wert"`.
Der Apache-Server nimmt die Anfrage entgegen, führt das Script aus und gibt ein HTML-Dokument zurück, das ggf. die Ausgaben des Scriptes enthält (bei Leseanfragen).
Das Dokument kann als **InputStream** von der Instanz von **URLConnection** eingelesen werden. Es wird dabei als String zeilenweise gelesen.
Mit dem Ende des Einlesens erfolgt der Aufruf der Methode **onPostExecute** im **AsyncTask**, die im Mainthread läuft und zur Rückgabe und Weiterverarbeitung der ausgelesenen Informationen gedacht ist. Das Problem dabei: die Antwort muss nun zurück in das Activity, das die Abfrage gestartet hat. Android selbst stellt dafür keine Lösung bereit.
Für diese Aufgabe wurde das Interface **AsyncResponse** erstellt. Es enthält die Methode **processFinish**, der ein String übergeben wird. HTTP_Connection enthält nun eine Instanz des Interfaces, die von außen gesetzt werden kann. In **onPostExecute** ruft sie nun **processFinish** der aggregierten Instanz auf und übergibt den Inhalt des HTML-Dokuments als String.
Das Interface muss von dem Activity implementiert werden, das die Antwort der Abfrage verarbeitet. Dabei wird die leere Standardmethode wie bei einem Listener überschrieben und die Antwort kann nun in der Methode weiter verarbeitet und z.B. in Oberflächenelemente geschrieben werden.

2.5 Der PHP-Server

Der PHP-Server läuft im Apache Webserver, der Teil von XAMPP ist, auf dem zentralen Laptop. Er kann nur im lokalen Netzwerk durch Anfragen (lediglich von den App-Instanzen) angesprochen werden. Der „htdocs“-Ordner von XAMPP enthält die erstellten Scripte, welche Anfragen der App verarbeiten. Diese müssen auf die Datenbank zugreifen können und benötigen dafür ihre Konfigurationswerte. Diese sind in einer separaten Datei gespeichert. Diese und die Scripte werden bei der Initialisierung durch das Hauptprogramm automatisch überschrieben.

2.5.1 Aufgabe

Der Server muss Anfragen der App entgegennehmen und Zugriffe auf die Datenbank ausführen. Anfragen werden dabei per HTTP-GET gestellt. Die ausgeführten Zugriffe können sowohl Schreib- als auch

Lesezugriffe sein. Bei einer Leseanfrage muss eine Antwort an die App gesendet werden. Dies geschieht per HTML-Seite, welche allgemein bei jeder POST- oder GET-Anfrage an den Client zurückgegeben wird und vom Script manipuliert werden kann.

2.5.2 Umsetzung des Zugriffes auf die Datenbank

Zuerst wird die Datei „Konfiguration.php“ mit dem Befehl „**require_once**“ eingelesen. Sie enthält die Konfiguration in Bezug auf Host, Port, User, Passwort und Datenbankname als PHP-Variablen. Danach wird der eigentliche Zugriff auf die Datenbank mittels der Methode „**mysqli_connect**“ realisiert, welche diese Werte entgegen nimmt. Dann wird der String, welcher die eigentliche Abfrage enthält, mittels des Verbindungsbefehls „**.**“ zusammengefügt. Übergebene Parameter in der Anfrage können per „**$_GET['Parametername']**“ eingebaut werden (z.B. die Nummer der Station). Schließlich wird die Abfrage mittels der Methode **„mysqli_query“** durchgeführt.
Ist eine Rückgabe nötig, wird diese nun erzeugt. Dafür geht das jeweilige Script zunächst die Antwort der Datenbank zeilenweise durch, liest die benötigten Werte aus ihren Spalten aus und schreibt sie auf die Antwortseite mittels des Befehls „**echo**“.

2.6 Technische Umsetzung der Synchronisation

Das Hauptprogramm ist in JavaFX geschrieben. Es kann unter Verwendung des Datenbanktreibers jdbc (externe JAR-Library) und dem java.sql-Package direkt auf die Datenbank zugreifen.
Es leert anfangs die Datenbank und legt die Tabellen an, die benötigt werden. Dann trägt es Namen und Startnummern in Tabelle „namen“ ein sowie nötige Attribute wie die Zahl der Messstationen sowie die Zuordnung der Messtore zu den Stationen in die Tabelle „allgemein“. Diese Daten sind nötigt für die App, um die Startnummernauswahl für Start/Stopp oder die Anmeldung an die einzelnen Messstation zu realisieren.
Die Android-Clients führen die App in ihren verschiedenen Activitys zum Starten, Stoppen oder auch Strafzeiten eintragen aus. Android erlaubt keinen direkten MySQL-Zugriff, aber die Verbindung zum Server über GET-Anfragen per WLAN über das Internetprotokoll und HTTPS ist möglich. Die App braucht dafür die IP-Adresse des Servers, welche vom Nutzer eingegeben werden muss.
Der PHP-Server kann Anfragen entgegennehmen und Abfragen in die Datenbank ausführen. Die App sendet also anfangs Anfragen an den Server, z.B., um die Zahl der Messstationen zu erfragen. Der Server nimmt die Anfrage entgegen und führt hier das Script „`Abfrage_Stationen`“ aus. Dieses sendet eine Anfrage an die Datenbank, welche als Resultat die Zahl der Messtationen liefert. Das PHP-Script gibt die Zahl auf die vom Server zurückgegebene HTML-Seite aus.
Die App liest diese Seite per InputStream ein, trennt sie (bei mehreren, vom Server per „|“ getrennten) Datensätzen per Methode **String.split** auf und verarbeitet sie, z.B., indem die vorhandenen Messstationen zur Auswahl in einen Spinner geladen werden.
Bei bestimmten Aktionen der App wie dem Start entfällt die Verarbeitung der Rückgabe (nicht benötigt).
Der PHP-Server übernimmt außerdem die Zeiterhebungen bei Start und Stopp. Er liest die Systemzeit des Laptops aus, sodass die Zeitdifferenzen korrekt gebildet werden. Würden die Smartphones die Zeiterhebung übernehmen, wäre dies zwar exakter (Verzögerungen bei Übertragung der Anfrage an den Server

beeinflussen die Zeiten nicht). Jedoch ergäben sich beträchtliche Fehler bei der Zeitdifferenzenbildung, wenn die Uhren der Smartphones und des Laptops nicht synchron sind.
Das Hauptprogramm erledigt nun die Zusammenführung, Anzeige und Auswertung der Daten der Datenbank und seiner lokalen Daten.
Dafür liest es anfangs Starter, Startnummern und Kategorien ein und bereitet die Datenbank auf die nächste Kategorien vor. Das Programm startet im vorgegebenen Intervall, standardmäßig jede Zehntelsekunde, die Zeitberechnungsprozedur. Diese ordnet ggf. Startzeiten von der App den Startern zu, markiert sie als gestartet und rechnet die Zeiten entsprechend um (PHP hat ein eigenes Timestampformat). Zielzeiten in der Datenbank werden gleich behandelt. Strafen aus der Datenbank werden im nächsten Schritt ausgelesen, aufsummiert und ausgegeben.
Das Programm erkennt automatisch das Laufende (wenn der letzte Starter im Ziel ist) und sichert ggf. die aktuellen Daten. Diese können aber weiterhin überschrieben werden. Danach bietet es die Möglichkeit an, den nächsten Lauf bzw. die nächste Kategorie zu starten.
Ist die letzte Kategorie abgeschlossen, speichert das Programm eine Excel-Datei mit der Auswertung.

2.7 Beschreibung des Ablaufs einer Verwendung des Softwaresystems

Zuallererst wird das Hauptprogramm „Kanu_s.a.M." auf dem zentralen Laptop gestartet. Es öffnet zunächst das ConfigWindow, welches Starter und weitere Konfigurationsdaten einliest. Die Starterdaten können aus einer Excel-Tabelle ausgelesen oder hier eingegeben werden.
Beim Klick auf Weiter oder dem Schließen des Fensters wird erfragt, welche Kategorie starten soll. Danach öffnet sich das Hauptfenster, welches die einzelnen Läufe verwaltet. Start/Stopp von Startern ist in diesem Fenster per Buttonklick möglich, ist aber unpräzise. Das Feature ist auch in der App enthalten.
Das Fenster zeigt jederzeit an, wer auf der Strecke ist oder sie bereits beendet hat, dessen aktuelle Zeit mit und ohne Strafen (über die Datenbank mit der App synchronisiert) und bei Startern, die im Ziel sind, die Gesamtzeit.
Wird dieses Hauptfenster angezeigt, ist der Start der „Kanu s.a.M. App" möglich. In diese muss zunächst die IP-Adresse des zentralen Laptops im lokalen Netzwerk eingegeben werden. Sie wird auf der GUI des Hauptprogramms angezeigt. Nach der Eingabe wird die Verbindung mit dem Button „Verbinden" bestätigt. Nun kann man das gewünschte Feature auswählen.
Wählt man „STARTEN", öffnet sich ein zweites Activity. In diesem werden alle Startnummern des aktuellen Laufs in einem Spinner angezeigt. Man wählt nun die Startnummer, welche die Startlinie passiert, im Spinner aus und tippt auf „STARTEN!". Stoppen funktioniert analog.
Zum Eintragen von Strafzeiten tippt man auf den Spinner im ersten Activity und wählt seine Messstation aus. Beim Klick auf „Wählen" öffnet sich ein neues Activity, welches dieses Feature realisiert. Dort wählt man zunächst die Startnummer im Spinner aus, die eine Strafzeit bekommen soll. In einer Tabelle wählt man die Strafzeiten für das in der linken Spalte angegebene Tor rechts im Spinner aus. Dieser enthält nur die Werte „0" (fehlerfreie Durchfahrt, der Standardwert), „2" (Torpfosten berührt) und „50" (Auslassen oder falsche Richtung). Die Strafe wird beim Klick auf „Zeit nehmen!" eingelesen.
Wählt man eine andere Startnummer im Spinner, werden die gewählten Strafzeiten alle auf 0 zurückgesetzt. Wenn der letzte Starter das Ziel erreicht, werden die Zeiten im Hauptprogramm gespeichert. Es

besteht die Möglichkeit, den nächsten Lauf zu starten. Start- und Zielzeiten sowie Strafen sind bis zum Start des nächsten Laufs veränderbar. Ist der nächste Lauf der Wiederholungslauf, wird gefragt, welche Starter welchen Lauf wiederholen. Sonst wird der reguläre zweite Lauf gestartet.
Tritt keiner an oder ist der Wiederholungslauf beendet, wird das ConfigWindow wieder aufgerufen. Es zeigt den Zustand vor Kategoriestart. Eine Eintragung über die App ist bis zum nächsten Kategoriestart nicht möglich.
Beim Klick auf weiter muss die nächste Kategorie aus den übrigen Kategorien ausgewählt werden. Die Verwendung der App ist dann wieder möglich. Es ist kein Neustart erforderlich, nur die aktuelle Anzeige muss durch das Kippen des Bildschirms aktualisieren oder das aktuelle Activity neu aufgerufen werden. Sind alle Kategorien gefahren, wird eine Excel-Datei mit der Auswertung am per **javafx.scene.FileChooser** gewählten Ort gespeichert.

2.8 Aufgetretene Probleme und ihre Lösungen

Bei der Verwendung des Softwaresystems treten gelegentlich nicht näher bestimmbare Netzwerkfehler auf, welche die HTTP-Übertragung zwischen den Smartphones und dem zentralen Laptop unterbrechen. Dies sind zufällige Fehler, welche in Rechnernetzen unvermeidbar sind und kaum verhindert werden können. Dies wurde dadurch verbessert, dass die Scripte bei erfolgreicher Eintragung eine Bestätigung auf die HTML-Seite ausgeben. Der AsyncTask wiederholt das Absenden der Anfragen, bis diese eintrifft oder zehn erfolglose Versuche gestartet wurden. Besonders bei Start und Stopp ergeben sich aber hier ggf. Verzögerungen bis zu mehreren Sekunden, bis die Verbindung wiederhergestellt wird. Dadurch entstehen Fehler in der Zeitmessung. Bei gutem WLAN-Empfang treten solche Fehler jedoch nur äußerst selten auf.
Ein weiteres potentielles Problem ist das der WLAN-Reichweite. Die Streckenlänge in Jena beträgt unter 200 m. Offizielle Rennstrecken sind aber bis 400 m lang. Die WLAN-n-Spezifikation gibt nur maximal 150 m Reichweite vor. Diese kann zwar im Freien ohne Weiteres erreicht werden. Jedoch beträgt die Reichweite der WLAN-Hotspots von Laptops meist deutlich weniger. Außerdem steht der zentrale Laptop wahrscheinlich im Zelt bei der Ziellinie, sodass die (beidseitig verfügbare) Reichweite nicht vollständig ausgenutzt werden kann. Abhilfe schaffen kann hier die Verwendung eines älteren WLAN-Routers oder WLAN-Repeaters, welcher eine höhere Stabilität und Reichweite bietet. Dieser kann (wenn die Stromversorgung es zulässt) näher der Mitte aller Geräte positioniert werden. Damit verbessert sich die Netzwerkabdeckung erheblich. Der Server kann auch per WLAN mit dem Router verbunden werden. Durch die Verwendung des Internetprotokolls müssen Laptop und Smartphones nur im selben Netzwerk sein, dieses muss nicht vom Laptop aufgebaut werden. Reicht auch dies nicht, kann ein WLAN-Verstärker an eine Antenne des Routers angeschlossen werden. Ein geeignetes Gerät ist etwa mit dem PowerMicrowave WIFIPA24508W verfügbar [16]. Reicht auch das nicht, können in den Messstationen einzelne akkubetriebene WLAN-Repeater eingesetzt werden, welche das Signal des Routers oder Laptops verstärken. Die Funktionalität eines WLAN-Repeaters kann auch durch Smartphones vom Typ Samsung Galaxy S7 oder S7 Edge erreicht werden, siehe hierzu [11]. Alternativ lässt sich ein tragbarer WLAN-Repeater wie der INSTAR IN-Route P52 verwenden [7].
Ein anderes auftretendes Problem ist, dass das Hauptprogramm teilweise nicht flüssig abläuft. Dies geschieht aber nur bei schwachen Laptops mit rechenintensiven Hintergrundaufgaben. Grund dafür ist,

dass der komplexe Zeitberechnungsalgorithmus mit mehreren Datenbankzugriffen jede Zehntelsekunde ausgeführt werden muss. Eine mögliche Lösung ist das manuelle Herunterstellen der Synchronisierungstaktrate. Für jeden solchen Zugriff muss der Datenbanktreiber jdbc verfügbar sein. Dieser liegt in einer Library, welche im Unterordner „lib“ im Programmordner liegen muss. Fehlt sie, kann das Programm nicht ablaufen. Die Library kann nicht mit in die JAR-Datei des Programms gepackt werden, da Java sie von dort nur als InputStream bitweise auslesen und somit nicht verwenden kann. Somit muss die Library im externen Ordner bleiben.
Ein weiteres Problem ist, dass XAMPP nur unter Windows automatisch vom Hauptprogramm gestartet werden kann. Eine Lösung für Linux und MacOS ist jedoch in Arbeit.
Im Zusammenhang damit steht das Problem, dass bei manchen XAMPP-Installationen der Autostart vom Apache Webserver aufgrund fehlender Berechtigungen oder Protokolle nicht funktioniert. Auf den meisten Geräten funktioniert er aber problemlos.
Bei der Entwicklung der App trat das Problem auf, dass nach einem Update der Google Repositories plötzlich die Spinner mit weißer Schrift auf weißem Hintergrund dargestellt wurden. Dieses wurde dadurch gelöst, dass ein ungeeignetes Android-Standarddesign durch ein eigenes Design überschrieben wurde, das schwarze Schrift erzwingt.
Ebenfalls ohne erkennbaren Grund stürzte die App nach einer Neukompilierung beim Versuch, Strafzeiten einzutragen, ab. Aus einem bis heute unklaren Grund war der Zugriff auf die Spinner und TextViews der Tabelle, in welcher die Strafzeiten an den einzelnen Toren ausgewählt werden, über deren Instanz nicht mehr möglich. Dies wurde gelöst, indem die Adressen der TextViews und Spinners in zwei Arrays gespeichert wurden, über die der Zugriff und das Auslesen der korrekten Werte wieder möglich war.
Änderte man in einer früheren Programmversion die Konfiguration der Datenbank, hatte die App keinen Zugriff mehr. Dies liegt daran, dass die PHP-Konfigurationsdatei mit den aktuellen Wertem überschrieben werden musste. Außerdem fehlten auf den Testrechnern oft wichtige Scripte. Diese waren nicht kopiert worden. Eine zusätzliche Funktion des Programms kopiert nun die Scripte automatisch in das korrekte XAMPP-Unterverzeichnis.
In frühen Versionen traten oft Probleme bei der Zeiterhebung und besonders der Bildung von Zeitdifferenzen auf. Dies wurde durch die Klasse java.util.Calendar gelöst, welche alle solchen Berechnungen automatisch vornimmt.
Generell gilt, dass die hauptsächlich verwendete Sprache Java in ihren Spezifikationen JavaFX und der Android-Variante sehr stabil ist. Durch die Fehlerbehandlung mit try-catch und generell das Fehlersystem mit Exceptions stürzt das Programm selbst bei schweren Fehlern nicht ab, sondern bricht nur die aktuelle Aktion ab. Auch PHP und MySQL sind sehr unempfindlich gegenüber Fehlern. Komplettabstürze der Software waren selten und die letzte Programmversion mit allen Verbesserungen lief so stabil, dass sie mehrere komplette Programmdurchläufe ohne Fehler durchführen konnte.

3 Zusammenfassung und Schluss

3.1 Umsetzung der Konzeption

Das entwickelte Softwaresystem „Kanu s.a.M." kann vollumfänglich zur Erfassung von Lauf- und Strafzeiten beim Kanuslalom verwendet werden. Das Feature Auswertung kann noch optimiert werden, aber ist in jedem Fall der mit Papierlisten in puncto Komfort weit überlegen.
Der Aufbau des Systems ist wie gefordert kostenlos und einfach möglich. An Hardware werden lediglich ein (ggf. auch älterer) Laptop sowie eine Reihe Android-Smartphones ab Android 2.3.3 (Gingerbread) benötigt. Erreicht der Hotspot des Laptops die nötige WLAN-Reichweite nicht, kann diese durch einen WLAN-Verstärker oder einen akkubetriebenen WLAN-Repeater aufgebessert werden. Auch der Einbau (eventuell vorhandener) älteren Router ist möglich.
An Software werden wie gefordert nur gratis verfügbare Programme vorausgesetzt. Für die Ausführung des Hauptprogramm ist eine aktuelle Java Runtime Environment erforderlich, welche mit Windows bis einschließlich XP kompatibel ist. Die finale Version wird ebenfalls als Java-Interpreter-unabhängiges .exe-Binary zur Verfügung gestellt werden. Dies ist unter JavaFX mit einer Modifikation des Build-Manifests möglich. Zudem wird XAMPP benötigt, welches ebenfalls ab Windows XP funktionsfähig ist. Es ist in wenigen Minuten installiert und nimmt die umständliche Konfiguration seiner Teilpakete selbstständig vor. Der Benutzer muss nur noch (einmalig) XAMPP so einstellen, dass bei seinem Start automatisch der Apache Webserver und MySQL gestartet werden. Außerdem muss anfangs im Datenbankenverwaltungsprogramm mindestens eine Datenbank angelegt werden, was in einer Minute getan ist. Java selbst kann dies nicht.
Das Programm verwaltet die Starter selbstständig und kann sie auf Wunsch direkt aus einer Excel-Datei einlesen. Die Starter werden den Kategorien automatisch zugeordnet, sodass alle Starter der gleichen Kategorie parallel starten. Die Kategorien sind dabei völlig frei benennbar. Ist ihr Name als Excel-Mappe nicht zulässig (kürzer als 1 oder länger als 31 Zeichen), wird automatisch ein anderer Name generiert.
Das Programm muss nur noch so bedient werden, dass die nächste Kategorie bzw. der nächste Lauf gestartet wird, dazwischen läuft es von selbst (deswegen semi-automatisch). Das Starten und Stoppen von Startern ist per App genauso wie das Eintragen von Strafzeiten möglich.
Das Programmende wird automatisch erkannt und es wird eine übersichtliche Auswertung in einer Excel-Datei vorgenommen. Diese kann nun in verschiedenen (auch kostenlosen) Programmen wie LibreOffice geöffnet und leicht ausgewertet werden.
Die korrekten Werte des aktuellen Laufs werden jederzeit im dafür vorgesehenen Fenster angezeigt, über einen Anschluss an einen Beamer könnten diese sogar den Zuschauern live gezeigt werden.
Das System erfordert keinen Internetzugriff, was es etwas sicherer gegenüber Manipulationen macht.
Alles in allem wurde die Konzeption vollständig umgesetzt. Das Problem kann durch das Programm gelöst werden. Damit ist bewiesen, dass ein (minimal ungenaueres als ein professionelles) Zeitenmesssystem für den Kanusport auch nahezu kostenfrei einrichtbar ist. Dieses System ist modular aufgebaut, sodass es relativ leicht um weitere Funktionen erweitert werden kann.

3.2 Ausblick

Eine gute Verbesserungsmöglichkeit wäre die, einen passiven Ablaufmodus zu schreiben. Das Programm würde in diesem lediglich den aktuellen Stand anzeigen, ihn aber nicht selbst manipulieren können. Dieser Ablauf könnte auch auf einem zweiten Laptop ausgeführt werden, der (in WLAN-Reichweite) bei den Zuschauern steht und die aktuellen Zeiten per Beamer projiziert. Neben dieser Auswertung am Wettbewerbsort können die Werte über die Datenbank auch ins Internet gestreamt werden.
Auch denkbar wäre eine bessere, detailliertere Auswertung. Dafür könnten die Laufwerte (z.B. in einer Excel-Tabelle) an ein Fremdprogramm übergeben werden, welches im Verein SV Schott zu diesem Zweck entwickelt wird.
Die Manipulierbarkeit des Systems kann ebenfalls noch verringert werden. Zusätzliche Authkeys für die Scripte, Passwörter zur Anmeldung an die einzelnen Stationen in der App (welche in der finalen Version auf Google Play verfügbar sein wird) und eine Verschlüsselung der Anfragen durch md5 zum Schutz vor Man-in-the-Middle-Attacks sind ebenfalls leicht realisierbar.
In der App könnte jeweils auch in den Spinnern der Strafzeitenwahl die aktuelle Strafzeitenkombination angezeigt werden.
Die Stabilität der App-Verbindung ließe sich durch die Wiederholung der Anfragen bis zu einer Bestätigung verbessern. Sollte doch einmal ein Fehler auftreten, wären verbesserte Fehlermeldungen und eine verbesserte interne Fehlerdiagnose und -verhinderung hilfreich. Für den Fall eines Komplettabsturzes gibt es noch den Wiederherstellungsmodus. Er speichert Protokolldateien auf Wunsch am angegebenen Ort und kann sie später wiederherstellen. Jedoch muss bei ihm aktuell eine komplette Kategorie wiederholt werden. Man könnte ihn so verbessern, dass nur ein angefangener Lauf wiederholt werden muss.
Um Fehler und Abstürze zu vermeiden wäre auch das Abfangen möglicher Eingabefehler hilfreich. Beispiele sind doppelte Startnummern oder solche, die keine Zahl sind.
Das Start-Stopp-Signal des Programms könnte zusätzlich zu Buttons und App auch per Lichtschranke gegeben werden, was die Präzision enorm erhöhen würde. Dafür wäre nur die Implementation eines Listeners nötig, der den Start-Stopp-Button des Hauptprogramms auslöst.
Statt eines klassischen Listeners könnte auch ein preisgünstiger Einplatinen-PC (z.B. Arduino mit WLAN-Modul) eine HTTP-Request an den Laptop senden. Dabei muss aber beachtet werden, dass Arduino und Laptop exakt synchrone Uhren haben. Durch eine Lichtschranke, einen Arduino und eine simple Webcam oder Digitalkamera ließe sich auch einfach ein Fotofinish schießen, was aber hier nicht zwingend nötig ist.

Glossar

Activity	Entsprechung des Programmfensters unter Android. Eine App kann aus einer oder mehreren Activities bestehen, die sich gegenseitig starten.
Apache POI	„POI ist Open-Source-Software, die Java-APIs zum Lesen und Schreiben von Dateien im Dateiformat von Microsoft Office wie z. B. Word und Excel bereitstellt. “[15]
Dalvik Virtual Machine	Von Google entwickelte Java-Laufzeitumgebung für Android-Apps. Angepasst an kleine, verschiedene Bildschirmgrößen, Skalierung von Inhalten und optimiertes Bugtracing.
GET-Anfragen	Anfragen an Scripte eines PHP-Servers. Parameter werden dabei in der Webadresse übergeben. Nach dem Namen des aufzurufenden Scriptes, etwa `http://192.168.1.234/Zeit_eintragen.php`, stehen die Parameter in der Form `?station=0&starter=1&strafe=50`.
htdocs-Ordner	„HyperText Documents “, Ordner in XAMPP. Dort liegen alle Skripte und HTML-Seiten, die über den Webserver aufgerufen werden können.
Hypertext Transfer Protocol	Netzwerkprotokoll, das für die Übertragung von Anfragen an Server und Antworten an Clients in Form von Hypertext (HTML) entwickelt wurde.
IP	Protokoll, das Anfragen und Antworten im World Wide Web zu bestimmten Servern oder Clients lenkt, die über eine IP-Adresse eindeutig identifiziert sind.
Java Virtual Machine	Laufzeitumgebung, in der Java-Programme ausgeführt werden. Simuliert eine Maschine in der Maschine, die sich in allen untestützten Plattformen nach innen gleich verhält. Somit laufen Java-Programme auf allen Systemen gleich. Zudem stabiler gegenüber Fehlern.
JavaFX	Java-Spezifikation, welche performanceoptimiert und besonders zurm schnellen und einfachen Anlegen von grafischen Benutzeroberflächen ausgelegt ist.
MariaDB	„freies, relationales Open-Source-Datenbankverwaltungssystem, das durch eine Abspaltung (Fork) aus MySQL entstanden ist“[13]
MySQL	Sprache, die zur Erzeugung und Anlegung von Datenbanken genutzt wird.

PHP	Abkürzung von Hypertext PreProcessor. Scriptsprache, die hauptsächlich auf Servern ausgeführt wird. Nimmt eine POST- oder GET-Request entgegen, verarbeitet sie z.B. mit einem Zugriff auf eine MySQL-Datenbank und gibt eine HTML-Seite an den anfragenden Client zurück.
UI-Thread/Mainthread	Thread, in dem die eigentlichen Activities der App ausgeführt werden. Jede App erhält bei Start einen solchen Thread zugewiesen. Seine Hauptaufgabe ist die Verknüpfung von Oberflächenelementen mit dem Programmcode, der ihren verschiedenen Events zugewiesen ist. Außerdem ist er für die Modifikation der Oberflächenelemente wie neue Texte in einem TextView verantwortlich.
XAMPP	„XAMPP ermöglicht das einfache Installieren und Konfigurieren des Webservers Apache mit der Datenbank MariaDB bzw. SQLite und den Skriptsprachen Perl und PHP (mit PEAR). “[14]

Literatur

[1] ARD Sportschau: *(ohne Titel)*. http://rio.sportschau.de/rio2016/kanu426_v-ardgalerie.jpg. Version: 08.01.2017 14:47

[2] Canopus49: *Wi-Fi icon in pale blue*. https://upload.wikimedia.org/wikipedia/commons/thumb/4/44/WIFI_icon.svg/2000px-WIFI_icon.svg.png. Version: 26.12.2016 17:53

[3] Google Inc.: *Android Robot*. https://upload.wikimedia.org/wikipedia/commons/thumb/d/db/Angry_robot.svg/2000px-Angry_robot.svg.png. Version: 26.12.2016 18:23

[4] IMAS: *Basissystem*. http://www.imas-sport.com/index.php/de/zeitmessung/basissystem. Version: 26.12.2016 15:53

[5] IMAS: *Komplettsystem*. http://www.imas-sport.com/index.php/de/zeitmessung/komplettsystem. Version: 26.12.2016 16:00

[6] IMAS: *Versionen im Vergleich*. http://www.imas-sport.com/index.php/de/zeitmessung. Version: 26.12.2016 17:54

[7] INSTAR GmbH: *IN-Route P52*. https://www.amazon.de/INSTAR-Route-P52-Powerbank-Ladestation/dp/B00V9MA8SW. Version: 04.01.2017 19:13

[8] Lockwood, Alex: *Why Ice Cream Sandwich Crashes your App*. http://www.androiddesignpatterns.com/2012/06/app-force-close-honeycomb-ics.html. Version: 28.12.2016 15:56

[9] Pixabay: *Android Smartphone Clipart*. https://cdn.pixabay.com/photo/2016/03/31/19/26/android-1295022_960_720.png. Version: 26.12.2016 17:48

[10] ProTime Sportservice GmbH: *Preisliste*. http://www.zeitmessung.de/preisliste.html. Version: 26.12.2016 16:53

[11] Schaap, Jonas: *Samsung Galaxy S7 als WLAN-Verstärker nutzen - so geht's*. http://www.usp-forum.de/artikel-ratgeber/13068-samsung-galaxy-s7-wlan-verst-rker-nutzen-so-geht-s.html. Version: 04.01.2017 18:36

[12] Walter, Stéphanie: *Mobile and desktop device template*. https://upload.wikimedia.org/wikipedia/commons/d/d4/Devicetemplates_laptop-01.png. Version: 26.12.2016 17:44

[13] Wikipedia: *MariaDB*. https://de.wikipedia.org/wiki/MariaDB. Version: 26.12.2016 18:33

[14] Wikipedia: *XAMPP*. https://de.wikipedia.org/wiki/XAMPP. Version: 26.12.2016 18:37

[15] WIKIPEDIA: *Apache POI*. https://de.wikipedia.org/wiki/Apache_POI. Version: 31.12.2016 17:41

[16] WLAN-SHOP24.DE: *PowerMicrowave WIFIPA24508W*. https://www.wlan-shop24.de/PowerMicrowave-WIFIPA24508W-24-GHz-WLAN-Verstaerker-max-8000mW-5dBi-Antenne. Version: 04.01.2017 18:30

Alle Quellen, die für die Softwareentwicklung relevant waren, sind im **@author**-Tag der Javadocs oder an verwendeter Stelle in Programmkommentaren gekennzeichnet.

4 Anhang

4.1 Screenshots

4.1.1 Hauptprogramm

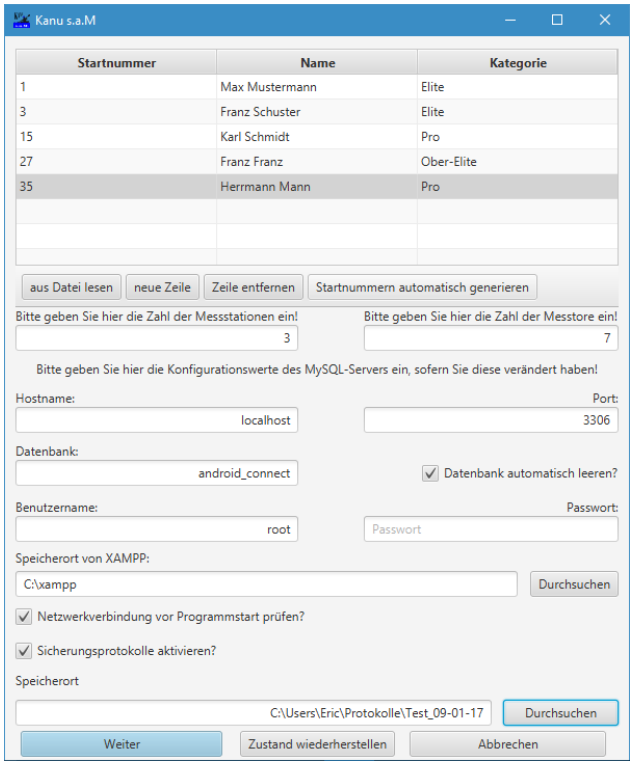


Abbildung 4.1: Konfigurationsfenster des Hauptprogramms mit Beispieleingaben

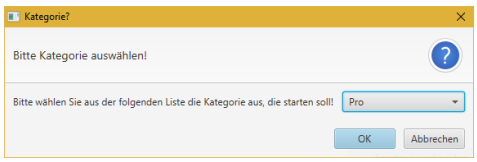


Abbildung 4.2: Wahl der Kategorie, welche starten soll

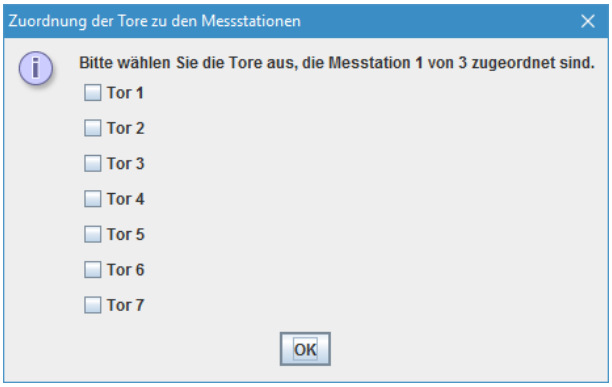


Abbildung 4.3: Zuordnung der Messtore zu Messstation 1

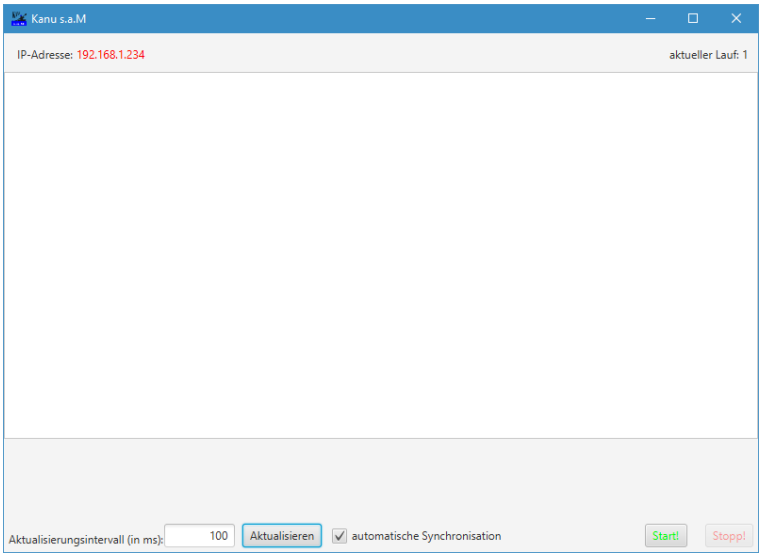


Abbildung 4.4: Hauptfenster vor Start des ersten Starters

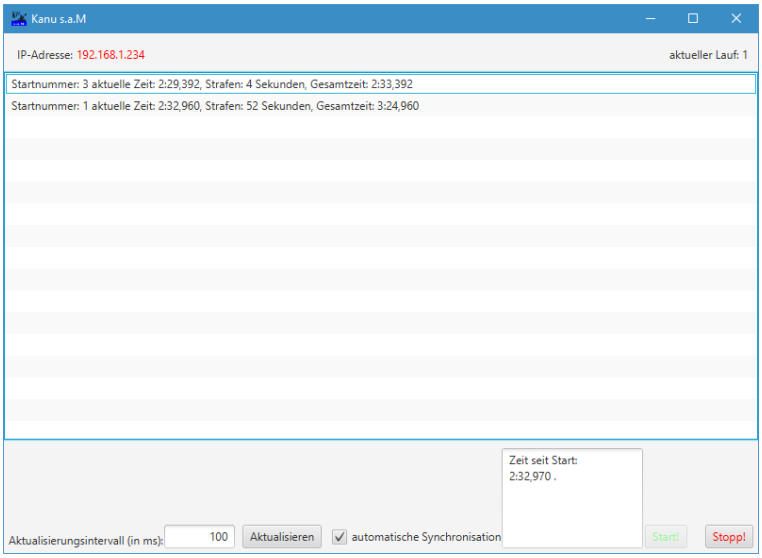


Abbildung 4.5: Hauptfenster während eines Laufs

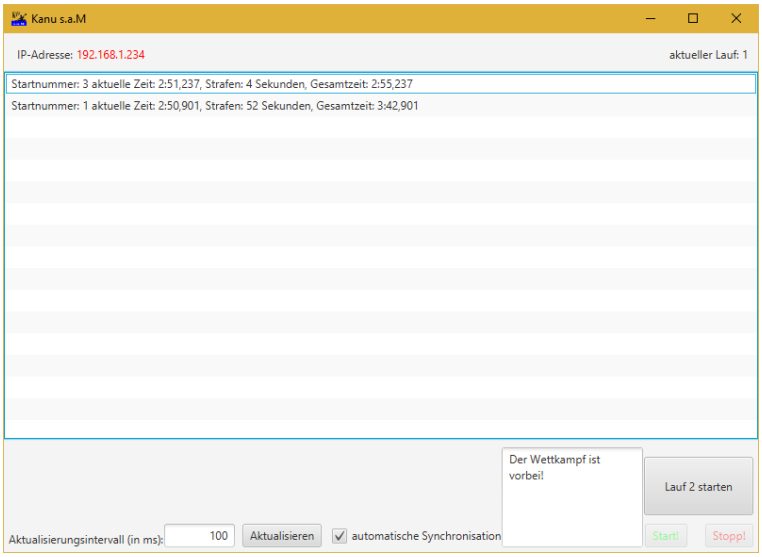


Abbildung 4.6: Hauptfenster bei Laufende

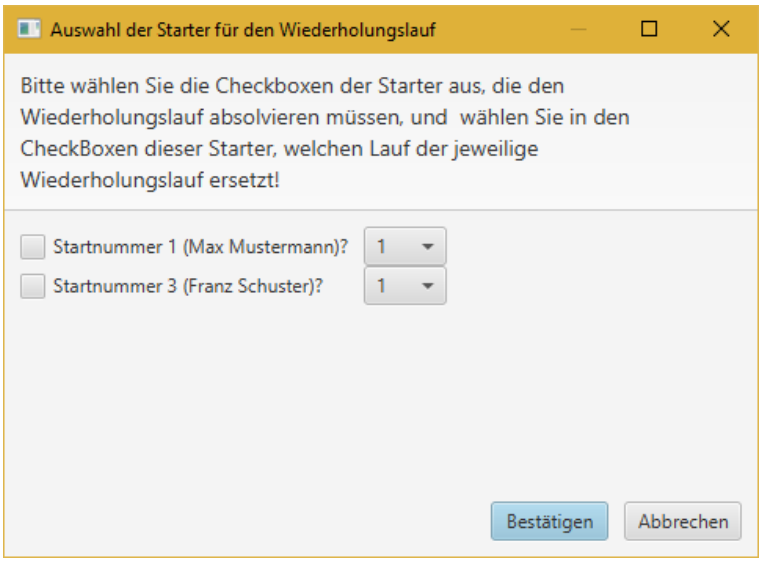


Abbildung 4.7: Auswahl der Starter, die zum Wiederholungslauf antreten

4.1.2 App

Abbildung 4.8: App vor dem Verbindungsaufbau

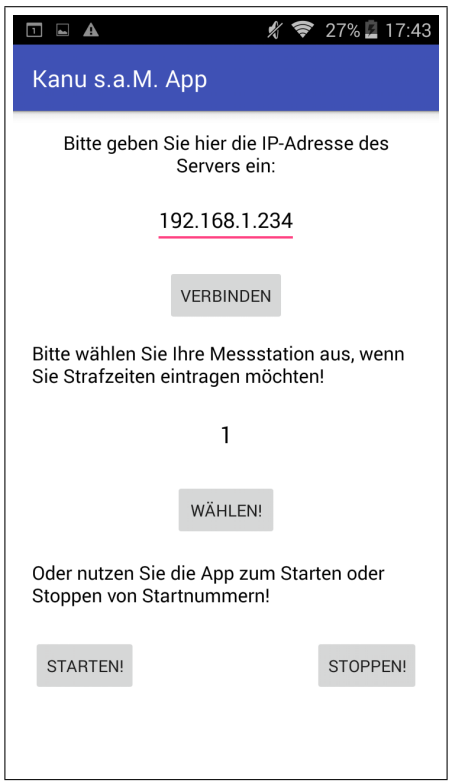


Abbildung 4.9: App bei der Funktionswahl

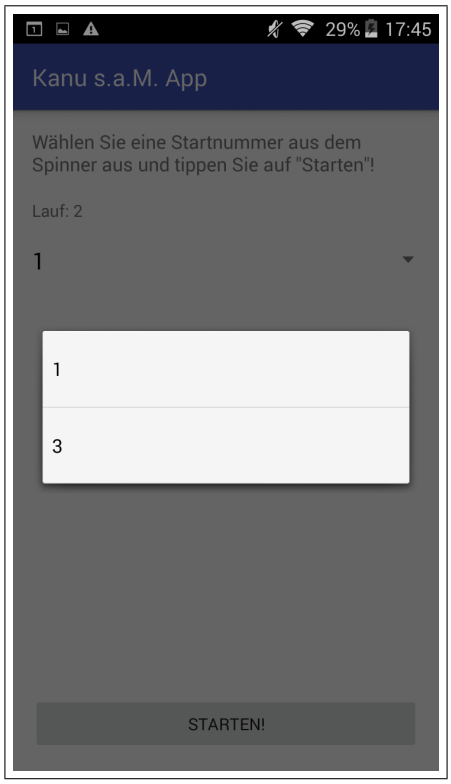


Abbildung 4.10: App im Startmodus bei Auswahl der Startnummer (Stoppmodus ähnlich)

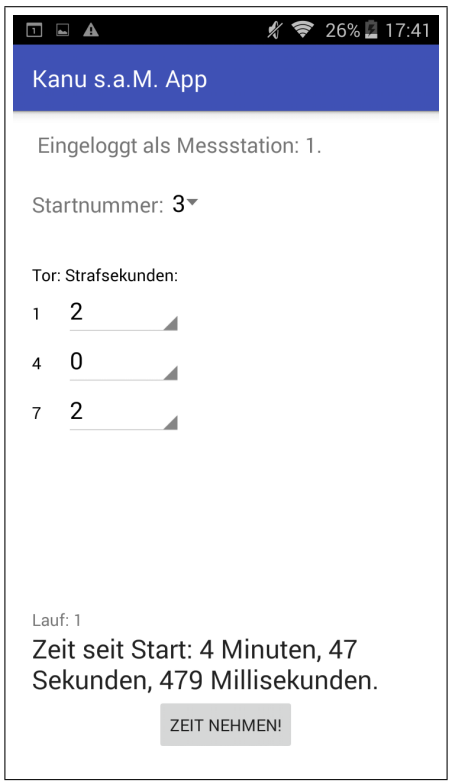


Abbildung 4.11: App beim Eintragen einer Strafzeit

4.1.3 Protokolle

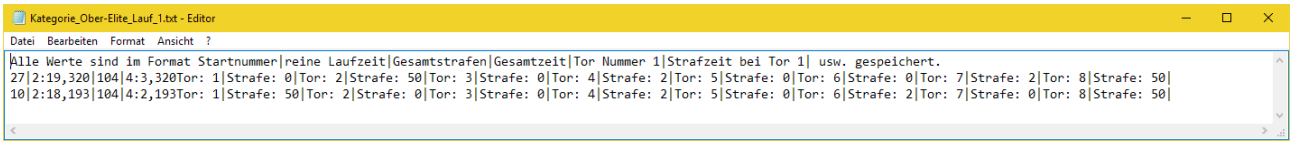


Abbildung 4.12: Protokoll eines Programmlaufs mit Beispielausgaben

4.2 Sourcecode

```
1  /**
2  * Hauptkonstruktor der Klasse
3  *
4  * @param host Datenbankhost (Standard: "localhost")
5  * @param port Datenbankport (Standard: "3306")
6  * @param db Datenbankname (Standard: "android_connect")
7  * @param user Datenbankbenutzername (Standard: "root")
8  * @param password Datenbankpasswort (Standard: leer)
9  */
10 MySQLConnection(String host, String port, String db, String user, String password,
       FXMLDocumentController doc) {
11         try {
12         //Verbindungsdaten speichern
13                 this.dbHost = host;
14                 this.dbPort = port;
15                 this.database = db;
```

```
16                  this.dbUser = user;
17                  this.dbPassword = password;
18                  // Datenbanktreiber für ODBC Schnittstellen aus der Library laden.
19                  //Resultat: im java.sql.DriverManager wird (statisch) der geladene Treiber
                        als Datenbanktreiber hinterlegt, dafür verantwortlich: Befehl im static-
                        Teil der Treiberdefinition, der bei Zugriff auf die Klasse ausgeführt
                        wird
20                  //Treiber wird nur gefunden, wenn lib-Unterordner da ist!
21                  Class.forName("com.mysql.jdbc.Driver");
22                  //Verbindung zur ODBC-Datenbank android_connect herstellen und für
                        Verwendung speichern.
23                  // Es wird die JDBC-ODBC-Brücke verwendet.
24                  conn = DriverManager.getConnection("jdbc:mysql://" + dbHost + ": + dbPort +
                        "/" + database + "?" + "user=" + dbUser + "&" + "password=" + dbPassword
                        );
25                  this.doc = doc;
26          } catch (ClassNotFoundException e) {
27                  //aufgerufen, wenn Treiber nicht gefunden wird
28          } catch (SQLException e) {
29                  //aufgerufen, wenn keine Verbindung zur DB möglich
30          }
31  }
```

Listing 1: Hauptkonstruktor der Klasse „MySQLConnection“, welche eine Verbindung zur Datenbank herstellt

```
1  //Konfiguration einlesen
2  require_once("Konfiguration.php");
3  //Datenbankverbindung aufbauen
4  $connection = mysqli_connect($host,$user,$passwort,$datenbank);
5  //Tore abfragen
6  selectTore($connection);
7
8  //liest alle der übergebenen Station zugeordneten Tore aus der Datenbank ein und gibt diese
       aus
9  function selectTore ($connection) {
10          //Abfrage zum Auslesen aller Tore
11          $sqlStmt = "SELECT Wert FROM `allgemein` WHERE `Attribut` = 'Messstation_".$_GET["
                station"]."_Tore'";
12          //Abfrage ausführen, Resultat in Variable result schreiben
13          $result =  mysqli_query($connection,$sqlStmt);
14          //falls Abfrage erfolgreich...
15          if ($result = $connection->query($sqlStmt)) {
16                  //für jede Zeile der Datenbankausgabe (nur 1 Zeile möglich, da Attribut Prim
                        ärschlüssel ist)...
17                  while ($row = $result->fetch_assoc()) {
18                          //die Tore ermitteln...
19                          $id = $row["Wert"];
20                          //und ausgeben.
21                          echo $id;
22                  }
23          //Ergebnisse leeren $result->free();
24          //Verbindung schließen
25          closeConnection($connection);
26          }
27  }
```

Listing 2: Methode „selectTore“des Scriptes „`Abfrage_Tore.php`“, welche alle Tore ermittelt, die der übergebenen Messstation zugeordnet sind.

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel „Wettkampferfassung und -auswertung beim Kanusport mittels eines kostenlos und einfach einrichtbaren Softwaresystems“ selbstständig, ohne unzulässige fremde Hilfe angefertigt und nur unter Verwendung der angegebenen Literatur und Hilfsmittel verfasst habe. Sämtliche Stellen, die wörtlich oder inhaltlich anderen Werken entnommen sind, wurden unter Angabe der Quellen als Entlehnung kenntlich gemacht. Dies trifft besonders auch auf Quellen aus dem Internet zu.

Eric Ackermann

Jena, 15.01.2017 Eric Ackermann