Tally
digitale Strichliste am Raspberry Pi

Nicolai Tegtmeier(Raspberry 4.1)
Dominik Scheffler(Website 4.3)
Philip Frieling(Tally 4.2)
Sebastian Reinke(Tally 4.2)

23.06.2015

Abbildung 1: Tally Logo

Inhaltsverzeichnis

1 Projektvorfeld

1.1 Kaffeestrichliste bisher

In kleineren Unternehmen und Arbeitsgruppen gibt es oft einen zentralen Kaffee-/ Getränkeautomaten und Snacks, an denen sich jeder Mitarbeiter bedienen kann. Um die Getränke und Snacks kaufen zu können wird meistens eine Strichliste auf Papier, in der Nähe des Automaten oder der Snacks angebracht, damit sich jeder Mitarbeiter eintragen kann, was er gekauft hat. Diese Liste wird oft zur besseren Verwaltung in eine Datenbank oder Tabelle eingepflegt, damit der Administrator einsehen kann wer wie viel gekauft hat. Hier passiert es jedoch häufig, dass die Strichliste sehr unleserlich wird und es bei der Übertragung in die Datenbank auf den PC zu Fehlern kommt. Außerdem ist diese Methode für den Verantwortlichen sehr zeitaufwendig, da alles per Hand übertragen werden muss.

Um diesem Problem entgegen zu wirken, erarbeiten wir eine neue Methode, um diese Daten 'digital' und somit einfacher speichern zu können.

2 Rahmenbedingungen

2.1 Aufgabe - die neue Kaffeestrichliste

Um eine möglichst energiesparende und handliche Lösung zu finden, sollte die neue Strichliste auf einem Raspberry Pi 2 realisiert werden. Zunächst soll eine passende Benutzeroberfläche für den Raspberry entwickelt werden, damit der Benutzer am Raspberry selber seine Einkäufe verbuchen kann. Dies soll mit Hilfe der integrierten Entwicklungsumgebung 'Qt Creator', die besonders zur Entwicklung von plattformunabhängigen C++ Programmen gedacht ist, entwickelt werden.

Mithilfe eines Webservers, der ebenfalls auf dem Raspberry läuft, soll die Administration von jedem Computer im selben Netzwerk über eine Weboberfläche möglich sein. Hier sollen auch die Benutzer ihre Daten einsehen und ändern können. Dazu wird eine MySQL/ SQLite Datenbank, sowie PHP Unterstützung benötigt. Mittels eines Barcodescanners soll es möglich sein am Raspberry Produkte ein zu scannen.

3 Ziele des Projekts

Die neue 'digitale Kaffeestrichliste' wird auf Basis eines Raspberry Pi 2 entwickelt, um die alte Strichliste abzulösen. Dazu wurden folgende zu erreichende Ziele festgelegt:

3.1 Zielbestimmung

3.1.1 Der Benutzer Account

Über den Benutzer Account soll der Benutzer sich am Raspberry selbst und an der Weboberfläche anmelden können. Am Raspberry selbst können Getränke und Snacks gekauft werden. Mithilfe der Weboberfläche kann der Benutzer Statistiken einsehen und sein Passwort ändern.

3.1.2 Das Programm

Das Programm das auf dem Raspberry läuft aktualisiert die Anzahl der Produkte im Lagerbestand automatisch, gibt Meldungen bei zu geringem Warebestand aus und gibt generelle Fehlermeldungen aus.

3.1.3 Der Administrator Account

Die Administration mithilfe des Administrator Accounts findet ausschliesslich über die Weboberfläche statt. Hier kann der Administrator Accounts hinzufügen und verwalten, Waren hinzufügen und verwalten und den Lagerbestand verwalten.

3.2 Produkteinsatz

3.2.1 Anwendungsbereiche

Mitarbeiter, beziehungsweise die Administratoren, können eine Strichliste 'digital' anlegen. Diese fasst den zu bezahlenden Betrag für die Mitarbeiter zusammen.

3.2.2 Zielgruppen

Personengruppen und Unternehmen bei denen Getränke und Snacks privat für alle angeboten und privat bezahlt werden, jedoch Schwierigkeiten mit der Handhabung einer herkömmlichen Strichliste haben und den Bestand an Getränken und Snacks erfassen wollen.

3.2.3 Betriebsbedingungen

Die Strichliste soll möglichst Wartungsarm und einen niedrigen Stromverbrauch haben. Desweiteren soll sie täglich vierundzwanzig Stunden laufen.

3.3 Produktumgebung

Das Produkt kann unabhängig an jedem Ort betrieben werden, solang eine Stromquelle in der Nähe ist.

3.3.1 Software

Die einzige Software die vom Benutzer benötigt wird ist ein aktueller Webbrowser.

3.3.2 Hardware

Der Benutzer kann mit jedem Internetfähigen Gerät die Weboberfläche erreichen.

3.3.3 Orgware

Der Administrator kann die Betriebsparameter konfigurieren.

3.4 Produktfunktion

3.4.1 Benutzerfunktion

Ein im System registrierter Benutzer kann das System erst nutzen, wenn er angemeldet ist. Nur ein Administrator kann einen Benutzer anlegen und ihm einen Benutzernamen sowie Passwort zuwesien. Sobald ein Benutzer registriert ist, kann er sich sowohl am Raspberry, als auch an der Weboberfläche anmelden. Dafür benötigt er seinen Benutzernamen und sein Passwort. Abmelden ist bei beiden Oberflächen jederzeit möglich.

Der Administrator kann über die Weboberfläche Produkte hinzufügen,entfernen und deren Details ändern. Benutzer können über Weboberfläche Favoriten einrichten und Statistiken einsehen.

Am Raspberry kann der registrierte und angemeldete Benutzer ein Produkt aus einer Liste wählen oder mithilfe eines Barcodescanners ein Prdoukt einscannen, welches in den Warenkorb gelegt wird. Desweiteren kann er die Anzahl der Produkte erhöhen und seinen gesamten Warenkorb einsehen. Wenn er alle Waren im Warenkorb hat, kann er zur Kasse und die Waren durch einen Klick bezahlen, wodurch sein Konto belastet wird. Ein Abbruch oder das erreichen des Hauptmenüs ist jederzeit möglich.

3.5 Programmfunktion

Beim Kauf eines Produktes aktualisiert das Programm automatisch den Warenbestand. Bei geringem Warenbestand wird eine Warnmeldung ausgegeben und bei fehlen des Artikels wird dieser entfernt.

3.6 Serverfunktion

Auf dem Raspberry soll ein Apache 2 Server mit PHP Unterstützung laufen.

3.6.1 Datenbank

Als Datenbank soll die resourcenschonendere SQLite Datenbank verwendet werden

3.7 Benutzerschnittstelle

Die Bedienung des Raspberry erfolgt über einen eingebauten Touchscreen am Raspberry. Außerdem wird ein Barcodescanner installiert um Produkte einscannen zu können.

4 Meilensteine des Projektes

4.1 Raspberry Pi 2 Model B - Die Hardware mit der passenden Software

Mit dem Raspberry Pi 2 war schon im Projektvorfeld eine passende Basis für das Projekt gegeben. Der Raspberry Pi 2 besitzt einen Vierkern ARM Cortex-A7 Prozessor mit jeweils 900MHz und einem Gigabyte Arbeitsspeicher. Desweiteren befinden sich vier USB 2.0 Ports, eine 40 GPIO pin Steckerleiste, ein HDMI Anschluss, ein Ethernet Port und ein Micro SD Kartenslot am Raspberry.

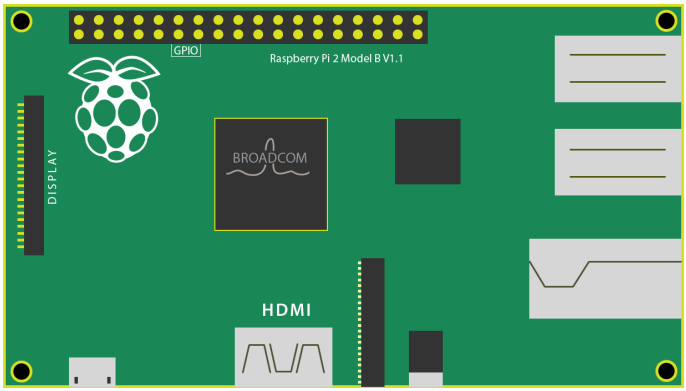

Abbildung 2: Der Raspberry Pi 2

Ausser dem Raspberry Pi 2 war zunächst ein 3,5 Zoll Touchscreen der Firma admatec [1] mit dem Namen C-Berry Touch vorhanden, das mittels eines Adapterboards an der GPIO Stiftleiste des Raspberry angesteckt wird. Das Display besitzt eine Auflösung von 320x240 Pixeln, eine LED- Hintergrundbeleuchtung und wird komplett vom Raspberry mit Strom versorgt.

4.1.1 Betriebssystem und Verbindung zu anderen Rechnern

Als erster Schritt wurde nach einem passenden Betriebssystem gesucht. Von Anfang an stand fest, dass ein Linuxderivat auf dem Raspberry laufen soll. Hier fiel die Wahl auf das speziell für den Raspberry entwickelte Raspbian.

'Raspbian' ist ein Debianderivat und wurde so optimiert, das es mit den geringen Hardwarespezifikationen zurecht kommt und den ARM-Prozessor des Raspberrys unterstützt. Als Alternative stand eine Ubuntu Version ,speziell für ARM-Prozessoren entwickelt, zu Verfügung, die jedoch noch nicht voll ausgereift ist und somit nicht einwandfrei und resourcenschonend auf dem Raspberry läuft.

Um die weitere Einrichtung des Raspberrys zu vereinfachen, wurde neben der vorkonfigurierten und installierten SSH Verbindung zur Verwaltung über ein Konsolenprogramm (Windows: Putty), eine freie Implementierung des Remote Desktop Protocols (Windows) für Linux Names 'xrdp' installiert. [2] .

```
1 sudo apt-get install xrdp
```

Damit ist eine einfache Remote Desktopverbindung zum Raspberry möglich und die komplette Verwaltung kann über einen Windows, Mac oder Linux Computer erfolgen.

4.1.2 Der Webserver

Als nächstes wurde ein vollständiger Webserver auf dem Raspberry eingerichtet. Hierzu wurde zunächst ein Apache Server der Version 2 installiert .

```
1 sudo apt-get install apache2
```

der über eine hohe Stabilität und Geschwindigkeit verfügt und serverseitig die Skriptsprache PHP unterstützt. Im Anschluss wurde das PHP5 Paket installiert .

```
1 sudo apt-get install php5
```

um eine volle Unterstützung für die kommende Website zu gewährleisten.

PHP ist eine Open Source-Skriptsprache, welche speziell für die Webprogrammierung geeignet ist. Bei einer Anfrage an die Website, wird der PHP-Code auf dem Server ausgeführt und erzeugt eine HTML-Ausgabe die dann letztendlich an den Client gesendet wird. [3]

4.1.3 Qt Creator

Das Programm an sich, Tally, wird auf einem externen Rechner in Qt programmiert und auf Richtigkeit überprüft, debuggt. Erst dann wird die Version auf GitHub zur Verfügung gestellt und auf den Raspberry geladen.

Damit das Programm auf dem Raspberry lauffähig gemacht werden kann, bzw kompiliert werden kann, muss der Qt Creator[4] samt Qt4 Umgebung installiert werden. Dieser kann dann das gespeichert unkompilierte Programm öffnen und passend für den Raspberry kompilieren.

Das Programm auf einem anderen Computer zu kompilieren entfällt, da das Programm speziell für ARM-Prozessoren kompiliert werden muss. Es gibt zwar die Möglichkeit des Cross-Compilierens, wo man auf einem anderen

System als dem gedachten das Programm schreibt aber so kompilieren kann das es auf dem Raspberry läuft. Diese weist aber noch einige Fehler beim kompilieren für den Raspberry auf, so dass ein fehlerfreies programmieren und kompilieren nicht möglich ist.

Für den Qt Creator mit den passenden development Tools führt man folgende Befehle aus .

```
sudo apt-get install qtcreator
sudo apt-get install qt4-dev-tools
sudo apt-get install gcc
sudo apt-get install xterm
sudo apt-get install git-core
sudo apt-get install subversion
```

Danach muss man nur noch die passende Toolchain, also den passenden Compiler für den Raspberry, auswählen. Dazu geht man im Qt Creator auf Optionen > build and run > tab tool chains > add und wählt GCC aus. Bei dem Compiler Pfad stellt man folgenden Pfad ein .

```
/usr/bin/arm-linux-gnueabihf-gcc-4.6
```

beim Debugger .

```
/usr/bin/gdb
```

Danach nur noch bestätigen und der Qt Creator ist einsatzbereit. Nun kann eine Projektdatei mit der Endung .pro geöffnet und kompiliert werden.

4.1.4 Die passende Datenbank

Nun stand die Auswahl des passenden Datenbankverwaltungssystems an. Zur Auswahl standen die Systeme MySQL und SQLite. Bei Recherchen zu den beiden Datenbanken stellte sich schnell heraus das die MySQL Datenbank zwar auf dem Raspberry lauffähig ist aber keine hohe Stabilität besitzt und sehr resourcenlastig auf dem Raspberry läuft. Daher wurde das SQLite Datenbanksystem ausgewählt. .

```
apt-get install sqlite3
apt-get install php5-sqlite
```

SQLite ist eine relationale Datenbank und benötigt im Gegensatz zu MySQL keine ständig laufende Software da die Datenbank aus einer einzigen, wenigen Kilobyte grossen, Datei besteht. SQLite legt Wert auf Geschwindigkeit und Einfachheit.

4.1.5 Der Touchscreen

Wie eingangs erwähnt stand neben dem Raspberry der C-Berry Touchscreen von admatec zur Verfügung. [1] Als erstes wurde der Treiber für den bereitgestellte C-Berry Touchscreen installiert [5] .

```
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.36.tar.gz
tar zxvf bcm2835-1.36.tar.gz
cd bcm2835-1.36
./configure
make
sudo make check
sudo make install
```

und im Anschluss die Beispielsoftware zum anzeigen von Bildern installiert .

```
wget http://admatec.de/sites/default/files/downloads/C-Berry.tar.gz
tar zxvf C-Berry.tar.gz
cd C-Berry/SW/tft_test
make
sudo ./tft_test
```

Nach ersten Tests mit dem C-Berry Touchscreen wurde bemerkt, dass es nicht als Primäres Display geeignet ist, da vom Desktop ein Screenshot erstellt wird und dieser auf dem Display ausgegeben wird. Starke Verzögerungen im Bildaufbau und eine schlechte Bedienung mithilfe des Touchscreens waren die Folge. Das Display ist dafür gedacht andere Inhalte als den Desktop wiederzugeben, die dann per Touchscreen bedient werden wie zum Beispiel einzelne Buttons.

Damit war der Touchscreen ungeeignet für das Projekt und ein neuer Touchscreen mit besserer Anbindung an den Rapsberry und einer höheren Bildwiederholungsrate musste gesucht werden. Dabei kam die Idee auf, ein 7 Zoll resistives Touchscreen von Pollin [6] zu verwenden, welches durch seine Grösse viel Platz für das Tally Programm bietet und eine sehr gute Touch Bedienung verspricht. Von Vorteil war auch die eigene externe Grafikeinheit die eine hohe Bildwiederholungsrateermöglicht und ein externer USB-Touchcontroller, der die Bedienung des Touchscreens besser ausführen soll.

Jedoch stellte die externe Stromversorgung ein Problem dar, da man jeweils ein Stromkabel für den Raspberry und eins für das Display benötigt, damit wäre eine Energiesparende Lösung ebenfalls nicht möglich.

Deshalb fiel die Entscheidung zugunsten des 3,5 Zoll Touchscreen 4DPi-35 von 4D System aus.[7] Der Touchscreen besitzt eine Auflösung von 480x320 Pixeln und durch die High Speed 48Mhz SPI Verbindung werden hohe und konstante Bildwiederholungsraten ermöglicht.

Abbildung 3: Der 4 DPi-35 Touchscreen

Die Vorteile bei diesem Display waren die kleine Bauform und keine zusätzlich benötigte Stromquelle. Ausserdem wird vom Hersteller direkt ein passender Kernel für Raspbian zur Verfügung gestellt. .

```
1 wget http://www.4dsystems.com.au /downloads/4DPi/kernel4dpi_1.3-3
      _pi2.deb
2
3 sudo dpkg -i kernel4dpi_1.3-3_pi2.deb
```

Nachdem das erforderliche Paket installiert wurde, musste der Raspberry herunter gefahren und vom Strom getrennt werden. Das Display wird an den GPIO-Ports angebracht und der Raspberry kann wieder mit Strom versorgt und hoch gefahren werden. Danach war ein sofortiger Betrieb möglich da der Raspberry das Display als primäres Display erkennt und direkt auf den Desktop bootet. Dadurch ist kein HDMI Bildschirm mehr notwendig und die Bedienung kann ausschliesslich über den Touchscreen erfolgen.

4.1.6 Produkte scannen

Zur vereinfachten Eingabe von Produkten soll ausserdem ein Barcodescanner angebracht werden. Hier sollte entweder ein handelsüblicher Barcodescanner per USB an den Raspberry angeschlossen werden, der Barcode mittels USB Webcam und passender Software oder aber ein Barcodescannermodul an den GPIO's des Raspberry als Lösung dienen.

Da jedoch das Barcodescannermodul nur mit ca. drei Volt betrieben werden kann, der Raspberry aber ca. fünf Volt an den GPIO's zur Verfügung stellt, fiel diese Möglichkeit weg. Deshalb war eine weitere Möglichkeit, eine Webcam an den Raspberry anzuschliessen und mittels einer Software das Bild auf Barcodes zu untersuchen, wobei es allerdings Probleme mit dem Autofokus

der Webcams gab. Damit ein Barcode gelesen werden kann, muss ein Autofokus vorhanden sein, der sich auch passend scharf stellt. Diese Webcams sind allerdings relativ teuer. Die Wahl fiel deshalb auf das Scannen mittels USB Barcodescanner, da dies eine einfacherere Handhabung verspricht und der Barcodescanner direkt als USB Tastatur erkannt wird.

4.1.7 Wlan für den Raspberry

Damit am Raspberry nicht immer ein Patchkabel angeschlossen werden muss und nur einen Stromanschluss benötigt wird, wurde der Edimax WLAN USB Stick[8] am Raspberry angeschlossen. Der Edimax WLAN Stick wird automatisch von Raspbian erkannt da schon alle erforderlichen Pakete und Treiber installiert sind. Daher wird der WLAN Stick empfohlen falls man WLAN am Raspberry nutzen möchte. Zunächst wurde der automatische 'Power Saving' Modus abgeschaltet, da der Wlan Stick sonst immer automatisch in den Ruhemodus geht und der Raspberry somit nicht mehr von aussen erreichbar wäre.

Hierzu wurde eine Konfigurationsdatei für den Treiber des WLAN Sticks erstellt. .

```
sudo nano /etc/modprobe.d/8192cu.conf
```

mit folgendem Inhalt .

```
options 8192cu rtw_power_mgnt=0 rtw_enusbss=0
```

Um nun eine Verbindung mit dem Wlan herzustellen wurde die folgende Datei .

```
sudo nano /etc/network/interfaces
```

um folgenden Inhalt ergänzt .

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-ap-scan 1
wpa-scan-ssid 1
wpa-ssid "WLAN-NAME"
wpa-psk "WLAN-PASSWORT"
```

damit eine Verbindung mit dem WLAN Netz hergestellt werden kann. Anschliessend muss nur noch per .

```
sudo service networking restart
```

der Netzwerkdienst neu gestartet werden und eine Verbindung mit dem angegebenen WLAN Netz wird hergestellt.

Da diese Möglichkeit jedoch relativ umständlich ist und Linux Kenntnisse erfordert, wird in das Tally Programm eine einfache WLAN Konfiguration eingebaut, in der man den WLAN-Namen und das WLAN-Passwort eingeben kann um sich mit dem WLAN-Netz zu verbinden.

4.1.8 angepasster Bootscreen

Während eines Meetings kam die Idee auf, den Bootscreen zu verändern und ein eigenes Bild während des hochfahrens anzuzeigen. Dazu wird eine weitere Software benötigt, der 'Frame Buffer Imageviewer', der ein Bild in den Speicherbuffer lädt, welches während des Hochfahrens angezeigt werden kann. [9]

Abbildung 4: Bootlogo

.

```
1 sudo apt-get install fbi
```

Danach muss ein passendes Skript geschrieben werden welches das Bild beim booten anzeigt. .

```
1 sudo nano asplashscreen
```

mit folgendem Inhalt .

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          asplashscreen
# Required-Start:
# Required-Stop:
# Should-Start:
# Default-Start:     S
# Default-Stop:
# Short-Description: Show custom splashscreen
# Description:       Show custom splashscreen
### END INIT INFO

do_start () {

    /usr/bin/fbi -T 1 -noverbose -a /etc/splash.png
    exit 0
}

case "$1" in
  start|"")
    do_start
    ;;
  restart|reload|force-reload)
    echo "Error: argument '$1' not supported" >&2
    exit 3
    ;;
  stop)
    # No-op
    ;;
  status)
    exit 0
    ;;
  *)
    echo "Usage: asplashscreen [start|stop]" >&2
    exit 3
    ;;
esac

:
```

Dieses Skript wird nun in den passenden Ordner verschoben damit es beim Start ausgeführt werden kann .

```
sudo mv asplashscreen /etc/init.d/asplashscreen
```

Als letztes wird das Skript für den Raspberry ausführbar gemacht und als Bootscript registriert .

```
sudo chmod a+x /etc/init.d/asplashscreen
sudo insserv /etc/init.d/asplashscreen
```

Nun wird w'ahrend des Bootvorgangs das Tally Logo angezeigt.

4.1.9 Backups einrichten und versenden per Mail

Als mögliche Backupvarianten standen ein komplettes Backup der Speicherkarte des Raspberrys oder ein Backup der Datenbank zur Wahl.

Der Entschluss fiel auf ein Backup der Datenbank, da diese auch per Mail an den jeweiligen Administrator geschickt werden soll. Dazu wird Rsync verwendet, das mit folgendem Befehl installiert wird: .

```
sudo apt-get install rsync
```

Rsync ist ein Programm, welches Dateien oder Ordner in verschiedenen Pfaden speichern kann. Dazu überprüft es zunächst ob die zu kopierende Datei geändert wurde oder nicht, wodurch ein unnützes kopieren von Daten unterbunden wird.

Damit die backups autmatisch und täglich erstellt werden, wird ein cronjob hinzugefügt. Cron-Daemon ist ein Dienst um Skripte oder Programme zu einer bestimmten Zeit starten zu lassen, ohne diese aktiv aufrufen zu müssen. Der Befehl dazu wird in der zugehörigen Tabelle, der crontab, gespeichert. Zunächst wird die Tabelle crontab mit dem editier-Befehl geöffnet .

```
sudo crontab -e
```

Hier wird folgende Zeile eingefügt .

```
15 22 * * * rsync -av --delete /var/www/sqlite/database.sqlite  /
    home/pi/Tally/sicherung
```

Dadurch wird das Programm jeden zum Beispiel Tag um 22:15Uhr aufgerufen und die Datenabank von /var/www/sqlite nach /home/pi/Tally/sicherung kopiert. Hier wurde die erste Stelle auf 15 gesetzt, welche die Minuten Angabe ist, die zweite auf 22 für die Stunden Angaben. Die dritte Stelle gibt an, an welchem Tag das Skript/Programm ausgeführt werden soll, die vierte an welchem Monat und die fünfte an welchem Wochentag. Da rsync täglich ausgeführt werden soll werden nur die ersten beiden Stellen angegeben.

Diese Datei soll nun auch täglich, nach dem erstellen des Backups, an den Admin per mail gesendet. Dafür wird das Programm sendEmail und zwei Perl- Libraries installiert. .

```
sudo apt-get install sendemail libio-socket-ssl-perl libnet-ssleay-
    perl
```

SendEmail hat den Vorteil, dass kein Mailserver auf dem Raspberry installiert werden muss.Der Raspberry wird als ein eigener 'sende Client' erkannt und nur die Zugangsdaten zum SMTP-Server des Email Accounts, über den gesendet werden soll, hinterlegt werden müssen. Die beiden Perl- Libraries dienen dazu, damit eine verschlüsselte Verbindung per TLS zum Mailserver aufgebaut werden kann.

Zum automatischen versenden der Mails wird folgendes Skript erstellt, welches vielfach für den Raspberry verwendet wird. Im Code findet sich der Name des Autors [10] .

```
sudo nano /usr/local/bin/mailnotify.sh
```

Das Skript sieht wie folgt aus .

```
################################################################################

               Sending E-Mail notification with sendEmail

 Creation:    15.08.2013
 Last Update: 20.10.2013

 Copyright (c) 2013 by Georg Kainzbauer <http://www.gtkdb.de>

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

################################################################################

#!/bin/bash

# Sender of the mail
SENDER="absender@domain.de"

# Recipient of the mail
RECIPIENT="empfaenger@domain.de"

# SMTP server
SMTPSERVER="smtp.domain.de"

# User name on the SMTP server
SMTPUSERNAME="MeinMailAccount"

# Password on the SMTP server
SMTPPASSWORD="MeinMailPasswort"

# Enable TLS for the SMTP connection
USETLS=1

#####################################################################
# NORMALLY THERE IS NO NEED TO CHANGE ANYTHING BELOW THIS COMMENT #
#####################################################################

# Use first argument as mail subject
if [ -n "$1" ]; then
  SUBJECT="$1"
else
  # No subject specified
  SUBJECT=""
fi

# Use second argument as mail body
if [ -n "$2" ]; then
  BODY="$2"
```

```
else
  # No mail body specified
  BODY=""
fi

# Generate the options list for sendEmail
OPTIONS=""

if [ -n "${SMTPSERVER}" ]; then
  OPTIONS="${OPTIONS} -s ${SMTPSERVER}"
fi

if [ -n "${SMTPUSERNAME}" ]; then
  OPTIONS="${OPTIONS} -xu ${SMTPUSERNAME}"
fi

if [ -n "${SMTPPASSWORD}" ]; then
  OPTIONS="${OPTIONS} -xp ${SMTPPASSWORD}"
fi

if [ -n "${USETLS}" ]; then
  if [ ${USETLS} == 1 ]; then
    OPTIONS="${OPTIONS} -o tls=yes"
  else
    OPTIONS="${OPTIONS} -o tls=no"
  fi
fi

# Send the mail with sendEmail
sendEmail -f ${SENDER} -t ${RECIPIENT} -u "${SUBJECT}" -m "${BODY}"
    ${OPTIONS}

exit 0
```

Zu editieren sind folgende Punkte: .

```
# Sender of the mail
SENDER="absender@domain.de"

# Recipient of the mail
RECIPIENT="empfaenger@domain.de"

# SMTP server
SMTPSERVER="smtp.domain.de"

# User name on the SMTP server
SMTPUSERNAME="MeinMailAccount"

# Password on the SMTP server
SMTPPASSWORD="MeinMailPasswort"
```

Hier müssen nur noch die sende Email Adresse, Sender of the mail, der Empfänger der Mail, Recipient of the mail und der SMTP Server mit dazugehörigen Benutzername und Passwort angegeben werden. Danach muss das Skript ausführbar gemacht werden .

```
sudo chmod +x /usr/local/bin/mailnotify.sh
```

Im Anschluss kann die erste Email versendet werden .

```
mailnotify.sh "Test" "Dies ist eine Testnachricht."
```

Damit wird das Skript aufgerufen, wobei der text in den ersten Anführungszeichen den Betreff, und der Text in den zweiten Anführungszeichen den Inhalt der Email enthält. Hierbei kam es jedoch zu folgendem Fehler .

```
invalid SSL_version specified at /usr/share/perl5/IO/Socket/SSL.pm
    line 332
```

Dieser Fehler entsteht, da die dazugehörige SSL Konfiguration einen kleinen Fehler enthählt. Folgende Datei muss aufgerufen werden .

```
sudo nano /usr/share/perl5/IO/Socket/SSL.pm
```

und in Zeile 1490 muss folgender Inhalt .

```
m{^(!?)(?:(SSL(?:v2|v3|v23|v2/3))|(TLSv1[12]?))$}i
```

durch .

```
m{^(!?)(?:(SSL(?:v2|v3|v23|v2/3))|(TLSv1[12]?))}i
```

ersetzt werden.Nur das Dollar Zeichen am Schluss muss entfernt werden. Anschliessend können Emails versendet werden .

```
mailnotify.sh "Test" "Dies ist eine Testnachricht."
Jul 15 16:32:30 raspberrypi sendEmail[4163]: Email was sent
    successfully!
```

Damit auch Anhänge mit versandt werden können muss lediglich das Skript etwas verändert werden. Hinzu kommt folgender Ausdruck. [11] .

```
# Use third argument as attachment
if [ -n "$3" ]; then
  ATTACHMENT="$3"
else
  # No attachment specified
  ATTACHMENT=""
fi
```

Somit sieht das Skript wie folgt aus .

```
#!/bin/bash

# Sender of the mail
SENDER="absender@domain.de"

# Recipient of the mail
RECIPIENT="empfaenger@domain.de"

# SMTP server
SMTPSERVER="smtp.domain.de"

# User name on the SMTP server
SMTPUSERNAME="MeinMailAccount"

# Password on the SMTP server
SMTPPASSWORD="MeinMailPasswort"

# Enable TLS for the SMTP connection
USETLS=1

######################################################################
# NORMALLY THERE IS NO NEED TO CHANGE ANYTHING BELOW THIS COMMENT #
######################################################################

# Use first argument as mail subject
if [ -n "$1" ]; then
  SUBJECT="$1"
else
  # No subject specified
  SUBJECT=""
fi

# Use second argument as mail body
if [ -n "$2" ]; then
  BODY="$2"
else
  # No mail body specified
  BODY=""
fi

# Use third argument as attachment
if [ -n "$3" ]; then
  ATTACHMENT="$3"
else
  # No attachment specified
  ATTACHMENT=""
fi

# Generate the options list for sendEmail
OPTIONS=""

if [ -n "${SMTPSERVER}" ]; then
  OPTIONS="${OPTIONS} -s ${SMTPSERVER}"
fi

if [ -n "${SMTPUSERNAME}" ]; then
  OPTIONS="${OPTIONS} -xu ${SMTPUSERNAME}"
fi

if [ -n "${SMTPPASSWORD}" ]; then
  OPTIONS="${OPTIONS} -xp ${SMTPPASSWORD}"
```

```
fi

if [ -n "${USETLS}" ]; then
  if [ ${USETLS} == 1 ]; then
    OPTIONS="${OPTIONS} -o tls=yes"
  else
    OPTIONS="${OPTIONS} -o tls=no"
  fi
fi

if [ -n "${ATTACHMENT}" ]; then
  OPTIONS="${OPTIONS} -a ${ATTACHMENT}"
fi

# Send the mail with sendEmail
sendEmail -f ${SENDER} -t ${RECIPIENT} -u "${SUBJECT}" -m "${BODY}"
    ${OPTIONS}

exit 0
```

Und in unserem Fall .

```
##############################################################################

               Sending E-Mail notification with sendEmail

 Creation:    15.08.2013
 Last Update: 06.04.2015

 Copyright (c) 2013-2015 by Georg Kainzbauer <http://www.gtkdb.de>

 This program is free software; you can redistribute it and/or
     modify
 it under the terms of the GNU General Public License as published
     by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

##############################################################################

#!/bin/bash

# Sender of the mail
SENDER="tallyupb@gmail.com"

# Recipient of the mail
RECIPIENT="tallyupb@gmail.com"

# SMTP server
SMTPSERVER="smtp.gmail.com"

# User name on the SMTP server
SMTPUSERNAME="tallyupb@gmail.com"

# Password on the SMTP server
SMTPPASSWORD="kaffee123"

# Enable TLS for the SMTP connection
USETLS=1

#################################################################
```

```
37 # NORMALLY THERE IS NO NEED TO CHANGE ANYTHING BELOW THIS COMMENT #
38 ####################################################################
39
40 # Use first argument as mail subject
41 if [ -n "$1" ]; then
42   SUBJECT="$1"
43 else
44   # No subject specified
45   SUBJECT=""
46 fi
47
48 # Use second argument as mail body
49 if [ -n "$2" ]; then
50   BODY="$2"
51 else
52   # No mail body specified
53   BODY=""
54 fi
55
56 # Use third argument as attachment
57 if [ -n "$3" ]; then
58  ATTACHMENT="$3"
59 else
60   # No attachment specified
61   ATTACHMENT=""
62 fi
63
64 # Generate the options list for sendEmail
65 OPTIONS=""
66
67 if [ -n "${SMTPSERVER}" ]; then
68   OPTIONS="${OPTIONS} -s ${SMTPSERVER}"
69 fi
70
71 if [ -n "${SMTPUSERNAME}" ]; then
72   OPTIONS="${OPTIONS} -xu ${SMTPUSERNAME}"
73 fi
74
75 if [ -n "${SMTPPASSWORD}" ]; then
76   OPTIONS="${OPTIONS} -xp ${SMTPPASSWORD}"
77 fi
78
79 if [ -n "${USETLS}" ]; then
80   if [ ${USETLS} == 1 ]; then
81     OPTIONS="${OPTIONS} -o tls=yes"
82   else
83     OPTIONS="${OPTIONS} -o tls=no"
84   fi
85 fi
86
87 if [ -n "${ATTACHMENT}" ]; then
88   OPTIONS="${OPTIONS} -a ${ATTACHMENT}"
89 fi
90
91 # Send the mail with sendEmail
92 sendEmail -f ${SENDER} -t ${RECIPIENT} -u "${SUBJECT}" -m "${BODY}"
       ${OPTIONS}
93
94 exit 0
```

Im Befehl zum senden wird nun ein drittes Argument hinzugefügt das den

Pfad mit der Datei, die zu versenden ist, enthält. .

```
mailnotify.sh "Test" "Dies ist eine Testnachricht." anhang.txt
```

Nun muss dieses Skript wieder in die crontab eingefügt werden, damit es täglich zu einer bestimmten Zeit ausgeführt wird .

```
sudo crontab -e
```

Hinzugefügt wird .

```
45 23 * * * /bin/bash /usr/local/bin/mailnotify.sh "Sicherung Tally
    Datenbank" "Dies ist eine automatische Sicherungsmail der Tally
    Datenbank" /home/pi/Tally/sicherung/database.sqlite
```

Damit wird das Skript mailnotify.sh um 23:45 Uhr gestartet und eine Email mit dem Betreff 'Sicherung Tally Datenbank', dem Inhalt der Mail 'Dies ist eine automatische Sicherungsmail der Tally Datenbank' und dem Dateianhang '/home/pi/Tally/sicherung/database.sqlite' gesendet.

4.1.10 Tally autostarten

Damit nach dem Starten des Raspberry Pi direkt das Tally Programm geladen wird, musste der Autostart für das Programm konfiguriert werden.

Die erste Idee war, das Programm per cronjob nach jedem neuen Hochfahren zu starten. Dafür musste zunächst ein kleines Skript geschrieben werden, dass das Programm startet, und dann in einer crontab nach jedem Hochfahren ausgeführt wird. .

```
sudo nano /etc/init.d/NameDesSkripts
```

Mit folgendem Inhalt .

```
#! /bin/sh
### BEGIN INIT INFO
# Provides: NAME DES PROGRAMMES
# Required-Start: $syslog
# Required-Stop: $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description:
# Description:
### END INIT INFO

case "$1" in
  start)
    echo "NAME DES PROGRAMMES wird gestartet"
    # Starte Programm
    /pfad/des/programmes
    ;;
  stop)
    echo "NAME DES PROGRAMMES wird beendet"
    # Beende Programm
    killall noip2
    ;;
  *)
    echo "Benutzt: /pfad/des/programmes {start|stop}"
    exit 1
    ;;
esac

exit 0
```

Danach wird das Skript ausführbar gemacht .

```
sudo chmod 755 /etc/init.d/NameDesSkripts
```

Um zu testen ob das Skript auch richtig funktioniert wird es mit folgendem Befehl aufgerufen .

```
sudo /etc/init.d/NameDesSkripts start
```

und wieder gestoppt .

```
sudo /etc/init.d/NameDesSkripts stop
```

Damit der Raspberry das Skript beim booten lädt wird noch folgender Befehl ausgeführt der das Programm in den passenden Runlevel bringt .

```
sudo update-rc.d NameDesSkripts defaults
```

Als nächstes soll das Skript durch einen im crontab hinterlegten Befehl jedes mal beim starten des Raspberrys ausgeführt werden .

```
sudo crontab -e
```

Dazu wurde folgende Zeile eingefügt .

```
@reboot \bin\bash /etc/init.d/NameDesSkripts
```

Das '@reboot' steht für das Ausführen der Datei nach jedem neuen Hochfahren des Raspberry. Jedoch startete das Programm nach dem Hochfahren des Raspberrys nicht. Erst als die Konsole geöffnet wurde startete das Skript und somit das Programm.

Die beschriebene Methode war lediglich dafür gedacht, Programme auszuführen, die keine GUI benötigen. Alle Programme, die Autostarten sollen und eine GUI benötigen, müssen über die globale LXDE (Lightweight X11 Desktop Environment) Autostart- Funktion, gestartet werden. [12] Dazu erstellt man in folgendem Pfad .

```
/etc/xdg/autostart/
```

eine Datei, die wie folgt lauten muss: 'NamedesProgramm.desktop'. Wichtig ist das die Datei auf '.desktop' endet, da sonst das Programm nicht nach dem hochfahren gestartet wird. .

```
[Desktop Entry]
Name=NamedesProgramm
Comment=
Exec=PfaddesProgramm -forever -rfbport 5900 -rfbauth ~/.vnc/x11vnc.
    pass -o ~/.vnc/x11vnc.log -display :0
Terminal=false
Type=Application
```

Diese Datei wieder ausführbar machen .

```
chmod +x x11vnc.desktop
```

Und das gewünschte Programm startet, nachdem die GUI geladen ist.

4.2 QT Creator - Das Tally Programm

Als Programmierumgebung, für den Raspberry Pi 2, wurde Qt gewählt. [13] Qt ist eine Klassenbibliothek in C++, vereinfacht die Programmierung für graphische Benutzeroberflächen und bietet zahlreiche weitere Funktionen an. Wir arbeiten mit der Version 4.8.1 von Qt, da neuere Versionen noch nicht mit dem Raspberry Pi 2 kompatibel ist. Als Debugger haben wir GNU gdb 7.8 und MinGW 4.9.1 als Compiler gewählt.

Programmiert wird hauptsächlich auf unseren eigenen PCs, da diese schneller debuggen können. Darüber hinaus musste der Raspberry PI 2 noch konfiguriert werden, wie in 4.1 beschrieben, wodurch ein zeitgleiches arbeiten erschwert wäre. Es wurde sich darauf geeinigt das Programm zu Demonstrationszwecke, während den Teammeetings, auf dem Raspberry PI 2 laufen zu lassen. Zum Ende des Projekts, sobald die Konfiguration beendet ist, sollten wir denn Raspberry PI 2 bekommen um die Benutzeroberfläche letztendlich anpassen zu können.

4.2.1 Die Benutzeroberfläche

Als erstes haben wir erste Grundzüge der Benutzeroberfläche thematisiert. Letztendlich haben wir die GUI herausgearbeitet, welche in dem Plichtenheft zu sehen ist.[Information] Außerdem wurden mehrere Strukturen für das Hauptprogramm besprochen und haben uns auf einen Automaten geeinigt. Näheres dazu findet sich in dem Unterpunkt Aufbau des Programmes.

Daraufhin wurde sich in Qt eingelesen und die ersten Funktionen für den Login-Screen programmiert. [14] [15] Wie bereits in Abschnitt 4.1 erklärt, wurde entschieden das Display zu wechseln. Außerdem ist uns Aufgefallen das sehr viele Klicks gebraucht werden um eine Kaufvorgang abzuschließen, was benutzerunfreundlich ist. Aus diesen beiden Gründen wurde die komplette Benutzeroberfläche überarbeitet.

Die Knopfgröße und die Schriftgröße mussten vergrößert werden als auch das Knöpfe, aufgrund Platzmangels, weggelassen werden mussten. Die Screens sollten aber erst beim Programmieren festgelegt werden um evtl. Hürden zu meistern. Außerdem wussten wir zu diesem Zeitpunkt noch nicht, welches Display für den Raspberry Pi 2 gewählt wird.

4.2.2 Datenbank und Tally

Nach dem die ersten Funktionen des Login-Screens und des Passwort-Screen implementiert wurden, wurde begonnen die Datenbank mit einzuarbeiten. Aus Abschnitt 4.1 geht heraus, dass wir uns für SQLite entschieden haben.

Um auf diese zuzugreifen, mussten wir uns zunächst darauf einigen, welche Tabellen angelegt werden und womit diese befüllt werden.

Nachdem wir uns auf die Datenbank geeinigt haben, wurde diese in das Programm implementiert. Zu diesem Zeitpunkt wurde noch nicht mit einer eindeutigen ID jedes Benutzer gearbeitet sondern mit seinem Namen, was später zu Komplikationen führen würde. Zeitgleich zum Login-Screen und dem Passwort-Screen wurden alle, die bis zu diesem Zeitpunkt erarbeitete Screen oberflächlich implementiert.

So konnten man zu diesem Zeitpunkt sich anmelden, (die Daten dafür wurden aus der Datenbank genommen) und konnte sich über alle Screens klicken, auch wenn die meisten keine Funktionen hatten.

Nachdem das Grundgerüst des Programmes stand, wurde um die Kauffunktion implementieren zu können, die ID der eingeloggten Person global im Programm bekannt gemacht werden. Dies haben wird entweder über get-Funktionen realisiert oder beim Erzeugen eines neuen Screens mitgegeben. Daraufhin konnten die restlichen Funktionen implementieren werden.

Als erstes wurden die Liste implementiert in welche alle kaufbaren Artikel aufgelistet werden. Die Information über die Artikel konnten über die Datenbank eingelesen werden. Nachdem sich die Listen mit Artikeln befüllten, wurde mit dem Hauptaspekt des Programmes begonnen, dem Kaufvorgang.

Für den Kaufvorgang ist hauptsächlich das Shoppingcart verantwortlich. Sobald ein Artikel dort liegt, wird es immer auf dem rechten Teil des Bildschirmes angezeigt. Außerdem ist es dafür verantwortlich, dass wenn Artikel gekauft werden, der Kaufvorgang korrekt abgeschlossen wird. Dies bedeutet vor allem, dass die Datenbank mit neuen Informationen gefüllt wird. Einerseits wird die Anzahl der entsprechenden Artikel geändert und das Konto des Käufers belastet. Dies bürgte eine große Hürde. Das Problem mit der Synchronität des Shoppingcart über den gesamten Einkauf.

Jedes Mal wenn der Zustannd gewechselt wurde(siehe Aufbau des Programmes), wurde das Shoppingcart gelöscht und damit auch der Inhalt. Das Shoppingcart ist ein eigenständiges Teil Programme und somit sollte es unabhängig sein, wenn sich der Zustand des Programmes ändert. Da wir allerdings keinen Weg gefunden haben ein Widget auf mehrere Widgets angezeigt zu haben, konnten wir das Problem auch nicht anders lösen. Das Soppingcart erstellt nun vor jedem löschen eine Liste mit seinen Items, welche dem neu erzeugten Shoppigcart gegeben wird.

Bis zu diesem Zeitpunkt wurde das Programm fast ausschließlich auf einem PC programmiert, debugt und auch die Größen wurden auf diesen angepasst, wie bereits anfangs erwähnt. Deswegen war der nächste Schritt die Benutzer Oberfläche, mit dem Raspberry PI 2 und dem neuen Display, anzupassen.

Zusätzlich wurde auch der Barcode Scanner implementiert, was kein Problem dargestellt hat, da sich das Signal des Barcode Scanners verhält wie ein Tastendruck mit der Tastatur. Qt stellt für solche Signale bereits Funktionen.

Somit war das Programm so gut wie fertig. Es fehlten nur noch ein paar kleine Funktionen und vor allem musste noch eventuelle Bugs entfernt werden. Alle Funktionen des Programmes wurden nicht während dieses Abschnittes erfasst, da dies den Umfang dieser Dokumentation sprengen würde. Alle wichtigen Funktionen wurde erwähnt und gegebenenfalls erläutert, darüber hinuas werden noch welche in dem Unterpunkt Aufbau des Programmes näheres erklärt.

4.2.3 Aufbau des Programmes

Das Programm wurde in 13 Klassen und einer main unterteilt.Im folgendem werden die einzelnen Klassen erläutert. Allgemein läuft das Programm nach einem Automaten ab. Der Bildschirm wurde in zwei Bereiche unterteilt, wodurch nicht bei jedem Zustandswechsel der komplette Bildschirm neu geladen werden muss, sondern nur der betreffende Teil. Dadurch Teilen wir den Bildschirm in die Kopfzeile und der Automatenbereich. Näheres dazu wird in den dementsprechenden Klassen erläutert. main.cpp : Die main.cpp ist verantwortlich für den Ablauf des Programmes. Die Struktur dafür ist aufgebaut wie ein Mealy-Automat.Die Zustände geben an welcher Bildschirm gerade angezeigt wird. Die Übergänge sind die Exitcodes. Die Ausgaben der Kanten sind jeweils der Screen-Wechsel, welcher nicht im Automaten abgebildet ist.

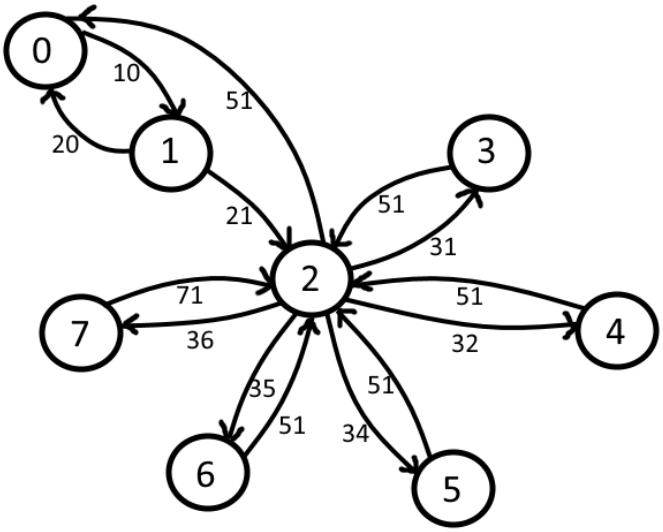

Abbildung 5: MEALEY

Exitcodes erlaubt es Programmen bei bestimmten Events(z.B. ein Knopf wurde geklickt) wieder zurück in die main.cpp zu gelangen. –>mainWindowPointer->exit(21); Es wird Anfangs die Ip ermittelt und diese auf einem eigenen Screen angezeigt. –>QTcpSocket socket; QString ip; .

```
1     socket.connectToHost('8.8.8.8', 53); // google DNS, or something
          else reliable
2     if (socket.waitForConnected()) {
3         ip = socket.localAddress().toString();
4     } else {
5         ip = socket.errorString();
6     }
7     w.showIpScreen(ip);
```

Zustände:
0 : Es wird der Login-Screen auf dem Bildschirm gezeigt. Exitcode:

1. 10 : Es wurde der Benutzer gewählt und möchte sich nun mit dem Account anmelden.
2. !10 & !100 & !34 : Das Programm beendet sich.

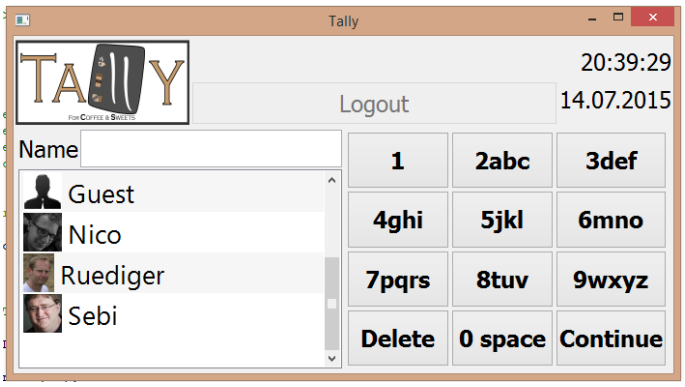

Abbildung 6: Login Screen

1 : Es wird der Passwort-Screen auf dem Bildschirm angezeigt. Exitcode:

1. 20 : Es wurde auf den 'Back' Knopf gedrückt gelangt in Zustand 0.
2. 21 : Das Passwort wurde korrekt eingegeben und man gelangt zum Coffee/Sweet/Scan Menü. !20 & !21 & !100 & !34 : Das Programm beendet sich.

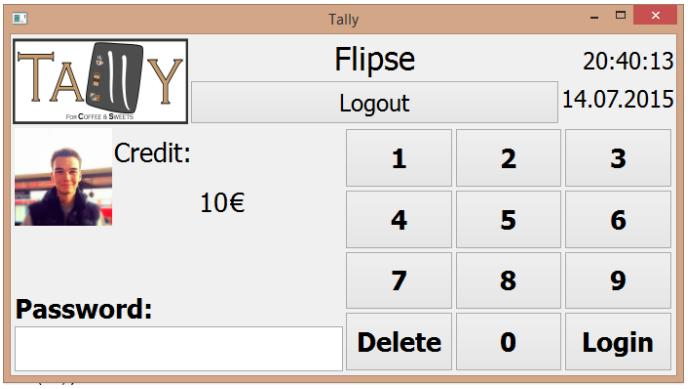

Abbildung 7: Passwort Screen

2 : Es gibt die Auswahlpunkte zwischen Coffee/Sweets/Scan als auch die Favoriten oder das Shoppingcart. Exitcode:

1. 31 : Es wurde auf 'Coffee' geklickt und man gelangt zur Getränkeliste mit dem Shoppingcart.
2. 32 : Es wurde auf 'Sweets' geklickt und man gelangt zur Süßigkeitenliste mit dem Shoppingcart.
3. 34 : Es wurde ein Artikel gescannt und wird in das Scan_Menue geleitet mit dem Shoppingcart.
4. 35 : Es wurde auf 'Favoriten' geklickt und man gelangt in den Favoriten-Screen.
5. 36 : Es wurde auf 'Settings' geklickt und wird in das Settingsmenue geschickt.
6. 51 : Es wurde auf 'Back' geklickt und man wird automatisch ausgeloggt.
7. 99 : Es wurde auf 'Buy' geklickt. Der Kaufvorgang wird beendet. Die Datenbank wird aktgualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-Screen. !31 & !32 & !33 ! & !34 & !99 & !100 & !98 : Das Programm beendet sich.

Abbildung 8: Item Screen

3 : Es wird die Getränkeliste mit dem Shoppingcart gezeigt. Exitcode:

1. 51 : Es wurde auf 'Back' geklickt und man wird in Zustand 2 gesetzt.
2. 99 : Es wurde auf 'Buy' geklickt. Der Kaufvorgang wird beendet. Die Datenbank wird aktgualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-Screen.
3. 98 : Ein Artikel wurde aus dem Warenkorb entfernt und wird wieder in der Getränkeliste aufgelistet.

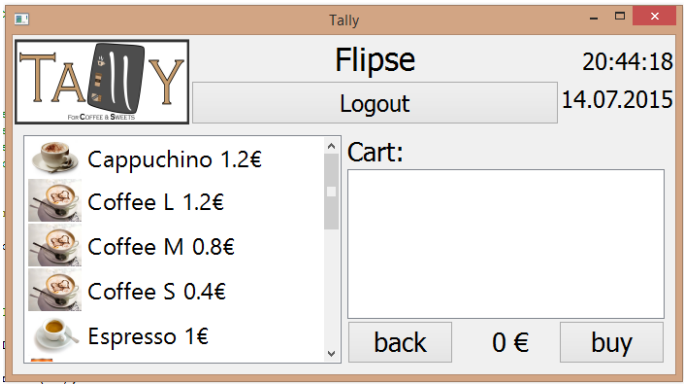

Abbildung 9: Coffe + Cart Screen

4 : Es wird die Süßigkeitenliste mit dem Shoppingcart gezeigt. Exitcode:

1. 51 : Es wurde auf 'Back' geklickt und man wird in Zustand 2 gesetzt.
2. 99 : Es wurde auf 'Buy' geklickt. Der Kaufvorgang wird beendet. Die Datenbank wird aktgualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-Screen.

3. 98 : Ein Artikel wurde aus dem Warenkorb entfernt und wird wieder in der Süßigkeitenliste angezeigt.

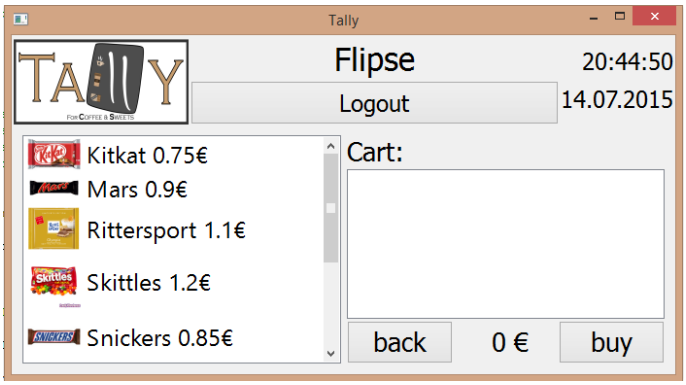

Abbildung 10: Login Screen

5 : Es wird das Scan-Menue angezeigt.

1. 51 : Es wurde auf 'Back' geklickt und man wird in Zustand 2 gesetzt.

2. 99 : Es wurde auf 'Buy' geklickt. Der Kaufvorgang wird beendet. Die Datenbank wird aktgualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-Screen.

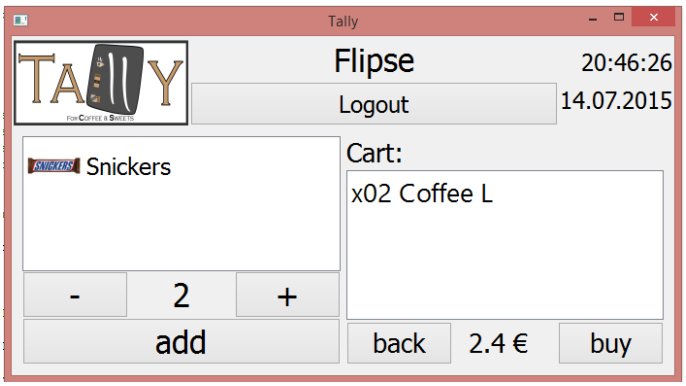

Abbildung 11: Scan Screen

6 : Es wird das Shoppingcart als auch die Favoriten angezeigt.

1. 99 : Es wurde auf ‘Buy’ geklickt. Der Kaufvorgang wird beendet. Die Datenbank wird aktgualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-Screen.
2. 51 : Es wurde auf ‘Back’ geklickt und man wird in Zustand 2 gesetzt.
3. 98 : Ein Artikel wurde aus dem Warenkorb entfernt und wird wieder in der Favoritenliste angezeigt.

7 : Es werden die Settings angezeigt(Nur als Admin)

1. 71 : Es wurde auf ‘Back’ geklickt und man wird in Zustand 2 gesetzt.

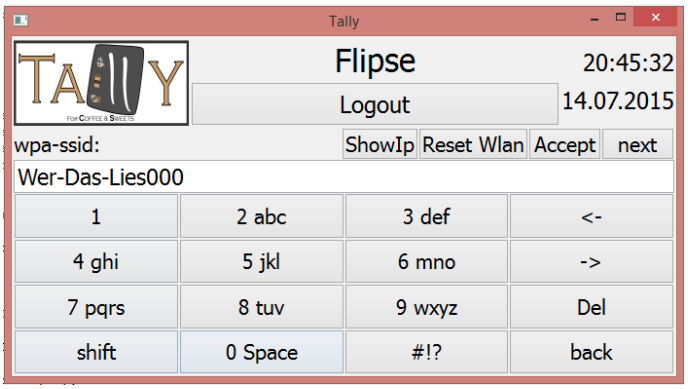

Abbildung 12: Settings Screen

mainwindows.cpp:
Wie bereits erwähnt haben wir die Oberfläche in zwei Teile unterteilt, in welche dann Widget hinzugefügt und entfertn werden können. Für diese Unterteilung ist die mainwindow.cpp verantwortlich, als auch noch für andere Funktionen.–>(BILD1)<– Die Kopfzeile wird erzeugt und dann nur noch verändert, bezüglich Datum,Zeit,Name und ob der Logout-Button aktiv ist. Der Automatenbereich wird mit jedem Zustanswechsel verändert. Durch die Funktion .

```
1 -->removeWidget()
```

werden alle Widgets die sich im Automatenbereich befindet entfernt, wodurch neue Widget dort gesetzt werden können. Fast jeder Screen besitzt eine Funktion names setMainWindowPointer. In diesem wird der Bezug zwischen dem Screen und dem mainwindow hergestellt. Außerdem wird dieser Funktion auch die User_Id mitübergeben, damit in der Klasse die User_id des angemeldeten Benutzers bekannt ist. .

```
1|        setMainWindowPointer(QApplication *a,QString gUserId)
```

Alle Funktionen die mit .

```
1|        show
```

beginnen, sind dafür da die entsprechenden Widget in dem Fenster zu erstmalig zu zeigen. Die Funktionen die mit .

```
1|        update
```

beginnen, updaten die entsprechend Widgets um auf entsprechende Events zu reagieren. Im Konstruktor der Klasse wird erstmalig die Uhrzeit und das Datum ermittelt. Dies funktioniert über die von Qt zur verfügung gestellten Klassen QTime und QDate. Diese Funktionen ermitteln die aktuelle Zeiten über das Betreibssystem, über welches Qt momentan läuft und gibt dieses in einem, vom Programmierer gewählten Format, zurück.

.

```
1|        QTime qtime = QTime::currentTime();
2|        QString stime = qtime.toString(Qt::LocalDate);
```

Zusätzlich bietet QTime noch die Funktion .

```
1|        timerEvent(QTimerEvent *event)
```

Diese Funktion wird jede Sekunde ausgeführt. Dadurch wird ein Zähler erzeugt denn wir für mehrere Funktionen genutzt haben. Einerseits wird die Zeit jede Sekunde aktualisiert um die aktuelle Uhrzeit auf der Oberfläche anzeigen zu lassen. Andererseits nutzten wir den Zähler auch für den Watchdog. Diese wird jede Sekunde einen hochgesetzt. Wenn eine gewisse Anzahl, welche in den Setting(Datenbank) angegeben ist, erreicht, wird der Benutzer automatisch abgemeldet und man landet wider in dem Login-Screen. Bei jedem Tastendruck, unabhängig in welchem Screen man sich befindet, wird der Watchdog wieder zurückgestellt.

Zusätzlich wird auch die, von Qt zur Verfügung gestellt Klasse QKeyEvent verwendet. In der Funktion .

```
1|        keyPressEvent(QKeyEvent *ev)
```

wird ein Tastendruck der Tastur erfasst und wird an einen String angehangen. Sobald 'Enter' gedrückt ist, wird ein Exitcode geworfen. Diese Funktion realisiert den Barcode-Scanner. Das Signals des Barcode_Scanners verhält sich wie ein Signal von der Tastatur und beendet den eingelesen Code mit einem 'Enter'. Mit dieser Funktion ist der Barcode_scanner implementiert.

loginscreen.cpp:
Im Login-Screen werden auf der Linken Seite alle Benutzer aufgelistet die in der Datenbank vorhanden sind. Dies geschieht über eine QListWidget in welche QListWidgetItem's hinzugefügt wird. In den QListWidgetItem's werden die Werte der Benutzer gespeichert, wlche über die Datenbank eingefortdert wurden. .

```
1  QListWidgetItem *item = new QListWidgetItem();      //
       Erzeugen eines Benutzers
2  item->setData(4,Database.getString(0).toInt());     //User\
       _Id wird an vierter Position gespeichert
3  item->setIcon(Database.getPixmap(3));               //Bild
       des Benutzers wird gespeichert
4  item->setText(Database.getString(2));               //
       Nickname des Benutzers wird gespeichert
5  ui->listWidget->addItem(item);                      //Neuer
       Benutzer wird in die Tabelle hinzugef\''ugt
```

Auf der rechten Seite erscheint eine Tastatur. Auf den 8 Knöpfen verteilt stehen die Buchstaben des Alpabhets. Jeweils 3 oder 4 Buchstaben pro Knopf. Nun kann man seinen Account in der Liste suchen. Entweder man scrollt in der Liste so lange bis man sich gefunden hat oder man gibt seinen Namen, rechts auf der Tastatur ein. Auf der Tastatur muss man seinem Namen mithilfe seines T9 Codes eingeben. Dies bedeuted jeder Buchstabe in seinem Namen wird durch eine Zahl ersetzt(Zahl laut Tastatur), wodurch die Namen auf der rechten Seite verschwinden, die keine Teil des bisher Eingegebenen Codes sind. Realisiert haben wir dies, indem von jedem Benutzer der T9_Code in der Datenbank gespeichert ist und nun mitHilfe des compare()Befehls von QString verglichen wird. .

```
1  if(tempID.contains(NameField) || NameField.length() == 0){
       //vergleicht den eingegeben Code(Namefield) mit den
        Werten aus der Datenbank
```

Wenn der Benutzer seinen Account gefunden hat kann er auf diesen klicken und gelangt automatisch in denn Passwort-Screen. Auserdem wird bei diesem Befehl, die User_Id gespeichert, womit andere Funktionen wissen welcher Bemnutzer sich anmelden möchte.

passwordscreen.cpp:
Im PasswordScreen erscheint auf der linken Seite, das Bild als auch der Credit des Benutzers. Auf der Rechten Seite befindet sich eine Tastatur mit den Zahlen von 0 bis 9, wodmit der Benutzer sich anmelden kann. Wenn der Benutzer sein Passwort korrekt eingegeben hat, wird bei der Datenbankabfrage bezüglich Benutzer und Passwort , den Wert true zurückgeben und man wird automatisch in den den Coffee/Sweet/Scan -Screen weitergeleitet. .

```
1  if(database.checkPassword(userId,password) \&\& blocked !=
       '1' \&\& database.checkUserLoginCount(userId)){  //
       Passwort wurde korrekt eingegeben und der User ist
       niocht geblockt
```

```
2             database.updateLoginAttempt(userId,true);
3             Data.close();
4             mainWindowPointer->exit(21);<--
```

Außerdem wurde der Zeitstempel für den User gesetzt. Falls sich der Benutzer drei mal mit dem Falschen Passwort anmelden wollte wird dieser Account für 60 Sekunden geblocket. Dies geschieht in dem wir, bei den letzten misslungten einloggen den Zeitstempel speichern und einen Zähler erhöhen(beide Werte werden in der Datenbank gespeichert). Nun wird bei jedem misslungenen einloggen der Zähler erhöht. Wenn dieser die drei erreicht, wird der account für 60 Sekunden gespeert. Sobald man sich einmal richtig einloggt wird der Zeitstempel aktualisiert und der Zähler wider auf 0 gesetzt. Die Funktion für die Datenbankänderung gibt die Klasse sqlzugriff, wobei die Abfrage über eine if-Anweisung geklärt wird.

.

```
1 if(database.checkPassword(userName,password))
```

Falls der Benutzer gesperrt ist, wird anstelle des Credits, die Nachricht angezeigt 'User blocked' angezeigt.

.

```
1         ui->label\_name->setText('User blocked');
```

Falls der Benutzer nur kurzeitig gespeert ist aufgrund zu häufiger Eingabe des falschen Passwortes, wird dieser auch geblockt. Dann erscheint ansttele des Credits 'Wrong login!' und 'Temp blocked.' Es wird eine Loginversuch immer dann als gescheitert definiert, wenn der Benutzer Login drückt und ein falsches Passwort eingegeben wurde und dann, wenn auf delete gedrückt wurde.

favwidget.cpp :
Diese Klasse beinhaltet den Screen der Favoriten.Diese enthält die Funktion setMainWindowPointer wodurch es die Schnittstelle liefert zwischen mainwindow.cpp und der favcart.cpp.

favcart.cpp & buywidget.cpp:
favcart.cpp zeigt die Favoriten an des jeweiligen Benutzers. buywidget.cpp zeigt entweder die Getränkeliste oder die Süßigkeitenlsite an. Welcher der beiden Liste gezeigt wird, wird dem der Funktion über die setMainwindow-Pointer Funktion bekanntgegeben.

.

```
1         setMainWindowPointer(QApplication *a,QList<QListWidgetItem>
              *cartItems,bool gSweetsActive,QString gUserId)
```

Wenn der bool Wert true ist wird die Süßigkeitenliste angezeigt, andernfalls die Getränkeliste. Die Werte dafür werden jeweils aus der Datenbank gezogen.

Beiden Klassen sind gleich Aufgebaut. Sie besitzen ein Tabelle wo alle Artikel aufgelistet sind. Dies wird über eine QListWidget organisiert wie bereits die Namen im Login-Screen. In den QListWidgetItem's sind folgende Werte an deren Position gespeichert:

.

```
1        - Der Name der Artikels wird der Name des QListWidgetItem's
             (item->setText(name);)
2        - Das Bild des Artikels hat einen extra Platz im
             QListWidgetItem (item->setIcon(picture);)
3        - An der vierten Stelle wird die Artikel ID gespeichert (
             item->setData(4,itmeID);)
4        - An der f\''unften Stelle wird der Preis des Artikels
             gespeichert (item->setData(5,price);)
5        - An der sechsten Stelle wird die Menge gespeichert die noch
              vorr\''atig ist (item->setData(5,amount);)
```

An den ersten drei Stellen sind wir uns nicht sicher was dort gespeichert wird. Wir können nichts darauf expliziet speichern und wenn wir ausgeben lassen was dort gespeichert ist, bekommen wir einen leeren String zurück.

Zusätzlich verändert sich die Farbe bei unterscheidlichen Beständen. Bei nur noch einem vorhanden Element wird der Name Rot angezeigt und bei unter 6 Elementen ändert sich die Farbe des Testes auf Gelb.

coffeesweetsscan.cpp :
In diesem Screen hat an die Möglichkeit zu wählen, zwischen der Getränkeliste und der Süßigkeitenliste. Gegebenenfalls bei Adminrechte auch den Button für die Settings. Diese Auswahlmoglichkeiten stehen auf der Rechten Seite und können durch anklicken ausgewählt werden. Auf der linken Seite befindet sich entweder das Shoppingcart, falls in diesem Items liegen oder falls das Shoppingcart leer ist, steht dort die Favoriten.

shoppingcart.cpp :
Das shoppingcart ist verantwortlich für den eigentlichen kauf. Es kombinbiert die Einkäufe aus den beiden Einkaufslisten, den eingescannten Artikeln und die aus dem Favoriten. Das Shoppingcart selber ist wider eine QListWidget mit QListWidgetItem's, wie bereits mehrmals verwendet. Mit der Funktion .

```
1        -->void Shoppingcart::updatePrice()
```

wird der Preis aller Artikel im Shoppingcart aktualisiert. .

```
1         while(ui->listWidget->item(loop) != NULL){
2        tempItem = ui->listWidget->item(loop);
3        price = price + (tempItem->data(5).toDouble() * tempItem->
             data(6).toInt());
4        loop++;
5     }
```

Falls der angemeldete Account nicht genügend Guthaben zur Verfügung steht wird der Preis rot markiert und es kann auch nicht eingekauft werden. Guthaben wird über die Datenbank ermittelt und in der Settings-Tabelle der Datenbank steht die Information um wie viel man sein Konto überziehen darf. Dies geschieht über die Funktion .

```
1         bool Shoppingcart::hasEnoughCredit(QString userId,double
              price)
```

Über die Funktion .

```
1         void Shoppingcart::addItem(QListWidgetItem *item)
```

wird ein Item in das Shoppingcart hinzugefügt.Außerdem wird Anzahl an ausgewählten Artikel auch in den Liste verringern. Dadurch synchonisieren sich die Listen, wodurch nicht mehr Artikel existieren als es eigentlich gibt. Die Funktion .

```
1         void Shoppingcart::on\_pushButton\_buy\_clicked()(Eingabe)
              <--
```

wickelt die kaufabwicklung ab. Diese öffnet die Datenbank. ändert den Credit des Benutzers

.

```
1         Database.updateCredits(userId,QString::number(credits.
              toDouble()-price));
```

Außerdem reduziert es die Artikelmenge in der Datenbank um die Anzahl, wie viele gekauft wurden und trägt die Käufe in den ConsumIndex-Tabelle, als auch in die Sell-History ein. .

```
1         Database.updateAmount(tempItem->data(4).toString(),QString::
              number(Database.getString(0).toInt() - tempItem->data(6)
              .toInt()));
2            Database.addSell(userId,tempItem->data(4).toString(),
                 tempItem->data(6).toString(),tempItem->data(5).
                 toString());
3        Database.updateConsumeIndex(userId,tempItem->data(4).toString
             (),tempItem->data(6).toString());
```

scanwidget.cpp :
Das Klasse wird erzeugt sobald ein Artikel eingescannt wurde, währrend

man in Zustand 2 sich befindet. Bei dieser MainWindowPointer Funktion wird noch zusätzlich der eingescannte Artikel, bereits als QListWidgetItem, mit übergeben, als auch das Shoppingcart. Der Eingescannte Artikel erscheit auf der linken Seite in einem QListWidget, wähhrend auf der rechten Seite das Shoppingcart angezeigt wird. Durch die Funktionen –>void ScanWidget::on_pushButton_plus_clicked() und –>void ScanWidget::on_pushButton_minus_clicked() wird die Anzahl des Artikel erhöht bzz verringert. Durch die Funktion .

```
1 void ScanWidget::updateAmountEveryItem()
```

wird die Farbe des Artikels geändert wie bereits zuvor in den Liste erwähnt. Durch die Funktion .

```
1         void ScanWidget::on\_pushButton\_add\_clicked()
```

wir der eingescannte Artikel in das Shoppigcart hinzugefügt. .

```
1         ui->listWidget->item(0)->setText(ui->listWidget->item(0)->
              text().insert(0,x0 + QString::number(count) +  ));
```

sqlzugriff.cpp :
Diese Klasse ist verantwortlich für die Datenbankzugriffe. Ein Zugriff funktioniert wie folgt: Man erzeugt zuerst eine Schlange vom Typ QSqlQuery . In diese werden später die gefunden Elemente aus der Datenbank gespeichert. Nun kann man mit der Funktion .exec auf bestimmte Teile der Datenbank zugreifen. .

```
1 QSqlQuery query;
2         query.exec('SELECT Value from Settings WHERE Setting\_ID =
              '3');
```

An der Stelle wo im Beispiel Settings steht, kommt der Name der Tabelle geschrieben, in welcher man suchen möchte. An der Stelle wo im Beispiel Value steht, kommt der Spaltenname geschrieben, welchen man suchen möchte. An der Stelle wo im Beispiel Setting_ID = '3' steht, steht eine Bedingung die die gesuchten Werte erfüllebn muss. Dieser Befehl muss nicht geschrieben wenn keine Bedingung vorhanden ist.

.

```
1         picPath = query.value(0);
```

Mit diesem Befehl, speichert man ein Element, welches in der Datenbank gefunden wurde. Falls es mehrer Werte geben sollte, benutzt man die Funktion –>query.next() Falls es ein nächstes Element existiert, wird das erste Elemnt der Schlange entfernt, wodurch das Element als nächstes ausgewertet werden kann. Falls kein Elemt mehr vorhanden ist, gibt die Funktion FALSE zurück.

Außerdem kann die Klasse auch in Datenbank schreiben. Einamla kann eine neue Splate erzeugt werden durch

.

```
1|        query.exec(INSERT INTO Consum\_Index (User\_ID, Grocery\_ID,
          |     Count) VALUES(+userId+,+Grocery\_Id+,+count+));
2|        Zuerst wird die Tabelle mit den Spalten, die gef\''ullt
          |    werden sollen genannt Consum\_Index (User\_ID, Grocery\
          |    _ID, Count) und dann werden die einzuf\''ugenden Werte
          |    genannt (+userId+,+Grocery\_Id+,+count+).
```

Um nun in einer Klasse auf die Datenbank zugreifen zu können, muss zuerst ein Objekt vom Typ QSqlDatabase erzeugt werden. Dann muss dem Objekt gesagt werden, das es mit einer SQL Datenbank arbeiten soll, auch die Stelle, an der man die Datenbank findet. .

```
1|        Data = QSqlDatabase::addDatabase(QSQLITE);
2|        Data.setDatabaseName(C:/SQLite/database.sqlite);
```

Letztendlich muus die Datenbank geöffnet werden durch .

```
1|        Data.open();
```

Ab nun kann man ein Objekt erzeugen von sqlzugriff und mit deren Funktionen auf die Datenbank zugreifen. Am Ende muss die Datenbank noch geschlossen werden durch .

```
1|        Data.close();
```

settingswidget.cpp :
In diesem Screen können die WLAN Einstellungen des Rapsberry Pi verändert werden. Der Admin kann dort die WLAN SSID und das Passwort über die extra dazu implementierte Tastatur GUI angeben/ändern. Klickt er dann auf 'accept' wird ein dirty Bit gesetzt, wodurch beim Verlassen des Widgets der WLAN Treiber neu gestartet wird. .

```
1|        process.start('sudo service networking restart');
```

afterbuyscreen.cpp :
Diese Klasse wird Aufgerufen, sobald der Warenkorb gekauft wurde. Sie enthält jediglich die Eigenschaft, mehrere Sekunden ein Screen zu zeigen auf dem der Benutzer erkennen kann, das der Kauf durchgeführt wurde. Danach wird diese wieder gelöscht und man gelangt in den Login-Screen.

showipscreen.cpp :
Diese Klasse wird nur beim Start des Programmes erzeugt. Sie dient dazu die IP-Adresse anzuzeigen.

4.3 Die Tally Website - Das Interface zur Einsicht und Konfiguration aller Daten

4.3.1 Erste Konzeption

Zur Kontrolle der Daten und aller Benutzerkonten haben wir uns entschieden, eine Web Applikation zu entwickeln, die Zugriff auf die Datenbank nehmen kann, um deren Informationen anzuzeigen / bearbeiten / löschen und ggf. neue Datensätze hinzufügen kann.

Zudem soll dies dadurch vereinfacht werden indem die Applikation für einige Einträge Vorschläge für Werte macht bzw. ausfiltert, falls ungültige Eingaben getätigt werden. Die App soll verschiedene Sicherheitsbereiche besitzen, die nur Erreichbar sind, falls ein Nutzer angemeldet ist.

Administrative Bereiche sollen bei gegebener Sicherheitsfreigabestufe (Eintrag 'Rights' in der 'Users' Tabelle der Datenbank) in die vorhandene Website eingebettet werden um unnötige Doppelungen von Bereichen zu vermeiden bzw. die Benutzung der App weiter zu erleichtern.

4.3.2 Ziele zur Bedienung und der Benutzeroberfläche

Der User soll ohne Anmeldung Zugriff auf die Bereiche 'Home' und 'Menu' haben. Dabei beinhaltet Home allgemeine Informationen und Neuigkeiten zur Seite, sowie sonstige Infos. Der Menübereich beinhaltet lediglich aktuelle Angaben zu verfügbaren Speisen ohne weiteren Zugriff in Form von Bestellungen oder Ähnliches.

Sobald sich der User angemeldet hat, verschwindet der Login-Button und wird von einem User-Dropdown abgelöst, in dem der User Links zu den Bereichen 'Notifications', 'Leaderboard', 'Settings', 'Favorites' und den Logout erhält. Es wird sichergestellt, dass der Client keinen Zugriff auf Daten bekommt bevor dieser die benötigten Userdaten zum Anmelden abgeschickt hat.

Im Notifications-Bereich kann der User seinen Kontostand, Nachrichten und letzte Einkäufe einsehen. Das Leaderboard dient nur Kosmetischen Unterhaltungszwecken und zeigt die Top 5 der Kilokalorien / Koffein Konsumenten, die dieses Feature aktiviert haben. In den Settings kann der User seine persönlichen Daten bearbeiten. Im Favorites-Tab kann er Favoriten aussuchen, die in der Schnellauswahl am Raspberry Pi angezeigt werden.

Der Logoutbereich entfernt alle gespeicherten Userdaten vom Client und trennt die aktive Sitzung. Der Admin hat folgende zusätzliche Features: Es werden im Menü Admin-Bedienelemente eingebettet, er hat einen separaten Bereich, um Userstatistiken einzusehen (Layout ist ähnlich dem Menü), wo er diese auch bearbeiten / löschen, oder neue User hinzufügen kann.

4.3.3 Umsetzung

Zu den Bereichen der Entwicklung gehören zum einen die Grundsprachen HTML5, CSS, JavaScript PHP und SQL, jedoch werden weitere Frameworks, Plugins und Erweiterungen benutzt, wie: jQuery-1.11.2, Mustache v2.1.2, Bootstrap-3.3.4, AngularJS v1.4.3-build.4096, Angular-UI-Router v0.2.15 und UI-Bootstrap-tpls-0.13.0.

In der ersten Version der Webapp, wurde ein Grundgerüst mit HTML5 CSS und Bootstrap erstellt. Der Fokus lag dabei beim Design und der Anordnung aller Elemente, die leichte Bedienbarkeit und eine selbsterklärende Grafische Benutzeroberfläche. Da der RaspberryPi sich möglicherweise innerhalb eines Intranetzes, ohne Internetzugriff befindet, muss dafür gesorgt werden dass die Libraries auch offline geladen werden können.

Hier ein Code-Auszug: .

```
<html ng-app='app'>
        <head>
                <title>Tally C&amp;amp;S</title>

                <!-- Latest compiled and locally saved CSS bootstrap
                     -->
                <link rel='stylesheet' href='styles/bootstrap-3.3.4-
                    dist/css/bootstrap.css'>

                <!-- Optional theme (locally saved) -->
                <link rel='stylesheet' href='styles/bootstrap-3.3.4-
                    dist/css/bootstrap-theme.css'>
        </head>
        <body class='color-bg'>
<!-- Plugins and Services -->
                        <!-- jQuery (necessary for Bootstrap's
                            JavaScript plugins. Saved on Drive) -->
                        <script src='scripts/jquery-1.11.2.js'></
                            script>
                        <!-- Latest compiled and locally saved
                            JavaScript from Bootstrap -->
                        <script src='styles/bootstrap-3.3.4-dist/js/
                            bootstrap.js'></script>
                        <!-- <script src='mustache/mustache.js'></
                            script> -->
                        <script src='angular/angular.js'></script>
                        <script src='angular-ui-router/build/angular
                            -ui-router.js'></script>
                        <script src='ui-bootstrap/ui-bootstrap-tpls
                            -0.13.0.js'></script>

                <!-- End Plugins and Services -->
        </body>
</html>
```

Um ein breiteres Spektrum zur DOM Traversal zu bekommen wurde jQuery in das Projekt eingebunden. Event-Listeners und Callbacks für z.B. Buttons sind somit einfach zu realisieren. Hier ist ein Codebeispiel für einen einfachen Button und ein Event-Listener in jQuery:

HTML Code: .

```
1         <button id='button1' class='btn btn-default'>This is a
              button</btn>
```

jQuery Code: .

```
1         $(function(){
2                 // DOM Element - Event  -  Callback
3                 $('#button1').on('click', function(){
4                         //Callback function
5                 });
6         });
```

Nach mehrfachen hinzufügen von HTML Seiten fiel auf, dass es zur ständigen Wiederholung einzelner Elemente wie z.B. der Navigationsleiste kommt, was bedeutet, dass man bei einer Änderung die selbe Codeänderung in jeder Datei durchführen muss.

Deshalb wurde das Mustache.js-Plugin zum Projekt hinzugefügt. Dies stellt einige Funktionen zum Templating von HTML Code zur Verfügung, welche den Code um einiges schrumpfen würde. Doch wurde Mustache nach kurzem Tests von Angular.js abgelöst, das teilweise die gleichen Funktionen wie Mustache besitzt (zum Thema Templating). Das größte Feature von Angular ist die Erstellung einer Single-Page-Webapp.

Anstelle vieler verschiedener HTML Dateien (für z.B. 'Home','Login','Menu') zwischen denen hin und her gesprungen wird, was ein ständiges nachladen aller Elemente bedeutet, bietet eine SPA den Vorteil das es nur eine Index.html Datei gibt in die eine app.js geladen wird und all ihre Komponenten. Von da aus lädt sich die App per Ajax (Asynchrones Java und XML) Templates, Controller, Services und Daten aus dem Backend dynamisch, während die Webapp geladen ist.

Die index.html wird somit zur Build Datei der App die die Komponenten der Webapp vereint und anschließend ausführt. Die Darstellung der Seite wird mit Directives zugewiesen, die sich in den Custom Tags von HTML5 befinden. Ein Beispiel für die Directives und dem Build der App in Codeform:

.

```
1         <html ng-app="app">
2                 <head>
3                         <title>Tally C&amp;amp;S</title>
4
5                         <!-- Laden der Stylesheets und anderes...-->
6
7                 </head>
8
9                 <body class="color-bg">
10
11                        <div id="wrap">
12
13                                <!-- Directive f\''ur die
                                      Navigationsleiste-->
```

```
14                          <my-navbar></my-navbar>
15
16                          <!-- Directive f\''ur das Login
                                ModalFenster-->
17                          <login></login>
18
19                          <!-- Directive f\''ur den Body oder
                                View der Seite-->
20                          <div ui-view></div>
21
22                      </div> <!-- close wrap -->
23
24                      <footer id="footer" class="text-center">
25
26                          <!-- Directive f\''ur die Fussleiste
                                -->
27                          <my-footer></my-footer>
28                      </footer>
29
30                      <!-- Plugins and Services -->
31                          <!-- jQuery (necessary for Bootstrap
                                's JavaScript plugins. Saved on
                                Drive) -->
32                          <script src="scripts/jquery-1.11.2.
                                js"></script>
33                          <!-- Latest compiled and locally
                                saved JavaScript from Bootstrap
                                -->
34                          <script src="styles/bootstrap-3.3.4-
                                dist/js/bootstrap.js"></script>
35                          <!-- <script src="mustache/mustache.
                                js"></script> -->
36                          <script src="angular/angular.js"></
                                script>
37                          <script src="angular-ui-router/build
                                /angular-ui-router.js"></script>
38                          <script src="ui-bootstrap/ui-
                                bootstrap-tpls-0.13.0.js"></
                                script>
39
40                      <!-- End Plugins and Services -->
41
42                      <!-- Build; Hier werden alle Elemente der
                            App geladen-->
43                          <script src="scripts/app.js"></
                                script>
44                          <script src="scripts/directives/
                                myNavbar.js"></script>
45                          <script src="scripts/directives/
                                login.js"></script>
46                          <script src="scripts/directives/
                                myFooter.js"></script>
47                          <script src="scripts/directives/
                                menuModal.js"></script>
48                          <script src="scripts/controllers/
                                userCtrl.js"></script>
49                          <script src="scripts/controllers/
                                menuCtrl.js"></script>
50                          <script src="scripts/controllers/
                                loginCtrl.js"></script>
51                          <script src="scripts/controllers/
                                loginModalCtrl.js"></script>
```

```
52                                  <script src="scripts/controllers/
                                        updatedCtrl.js"></script>
53                                  <script src="scripts/controllers/
                                        logoutCtrl.js"></script>
54                                  <script src="scripts/controllers/
                                        lbCtrl.js"></script>
55                                  <script src="scripts/services/
                                        myService.js"></script>
56                                  <script src="scripts/services/
                                        userDataService.js"></script>
57                                  <script src="scripts/services/
                                        menuService.js"></script>
58                                  <script src="scripts/services/
                                        loginService.js"></script>
59                                  <script src="scripts/services/
                                        loginModalService.js"></script>
60
61
62                          <!-- End Build -->
63
64                          <script src="scripts/main.js"></script>
65                  </body>
66          </html>
```

Zum Routing auf der Website wurde der Angular-UI-Router ins Projekt eingebunden. Er ermöglicht das Filtern vom User eingegebenen/ausgewählten Routen nach Authentifikation und Authorisierung und ungültigen Pfaden.

Innerhalb von app.js wird die App wie folgt gebaut:

.

```
1
2           var app = angular.module('app',[
3                   'ui.router',
4                   'ui.bootstrap',
5                   'app.directives.myNavbar',
6                   'app.directives.myFooter',
7                   'app.directives.login',
8                   'app.directives.menuModal',
9
10                  'app.controllers.userCtrl',
11                  'app.controllers.menuCtrl',
12                  'app.controllers.loginCtrl',
13                  'app.controllers.loginModalCtrl',
14                  'app.controllers.updatedCtrl',
15                  'app.controllers.logoutCtrl',
16                  'app.controllers.lbCtrl',
17
18                  'app.services.myService',
19                  'app.services.userDataService',
20                  'app.services.menuService',
21                  'app.services.loginService',
22                  'app.services.loginModalService'
23
24                  ]);
```

Dabei stellt alles, das sich in dem Array befindet, eine Dependency dar, die vor der Ausführung der App geladen werden muss.

Zum Routing auf der Website wurde der Angular-UI-Router ins Projekt eingebunden. Er ermöglicht das Filtern vom User eingegebenen/ausgewählten Routen nach Authentifikation und Authorisierung und ungültigen Pfaden. Wenn der User im Begriff ist einen Bereich zu betreten der eine Authentifikation benötigt wird überprüft ob Userdaten vorhanden sind mit denen sich der Client identifizieren kann. Falls nicht aka der Client nicht angemeldet ist wird kein Aufruf zum Backend gestartet, sondern der Client zum Home-Bildschirm weitergeleitet mit der Bitte, sich anzumelden. Hier der Code für das Routing:

```
1         //UI-ROUTER
2         app.config(['$urlRouterProvider','$stateProvider',function(
              $urlRouterProvider, $stateProvider) {
3                 $urlRouterProvider.otherwise('/');
4                 $stateProvider
5                         .state('/', {
6                                 url: '/',
7                                 templateUrl: 'ng-templates/home.html
                                      ',
8                                 data: {
9                                         requireLogin: false
10                                }
11                        })
12                        .state('home', {
13                                url: '/home',
14                                templateUrl: 'ng-templates/home.html
                                      ',
15                                data: {
16                                        requireLogin: false
17                                }
18                        })
19                        .state('menu', {
20                                url: '/menu',
21                                templateUrl: 'ng-templates/menu.html
                                      ',
22                                data: {
23                                        requireLogin: false
24                                }
25                        })
26                        .state('app' ,{
27                                abstract: true,
28                                data: {
29                                        requireLogin: true //this
                                              property will apply to
                                              all children of 'app'
30                                }
31                        })
32                        .state('notifications', {
33                                url: '/notifications',
34                                templateUrl: 'ng-templates/
                                      notifications.html',
35                                data: {
36                                        requireLogin: true
37                                }
38                        })
39                        .state('leaderboard', {
40                                url: '/leaderboard',
```

```
41                                  templateUrl: 'ng-templates/
                                        leaderboard.html',
42                                  data: {
43                                          requireLogin: true
44                                  }
45                          })
46                          .state('settings', {
47                                  url: '/settings',
48                                  templateUrl: 'ng-templates/settings.
                                        html',
49                                  data: {
50                                          requireLogin: true
51                                  }
52                          })
53                          .state('favorites', {
54                                  url: '/favorites',
55                                  templateUrl: 'ng-templates/favorites
                                        .html',
56                                  data: {
57                                          requireLogin: true
58                                  }
59                          })
60                          .state('logout', {
61                                  url: '/logout',
62                                  templateUrl: 'ng-templates/logout.
                                        html',
63                                  controller: 'logoutCtrl',
64                                  data: {
65                                          requireLogin: true
66                                  }
67                          })
68          }]);
```

URLs werden nicht als tatsächliche URL behandelt sondern in einen State verwandelt. Wenn der User also zur URL '/home' weitergeleitet wird, ändert sich der State zu 'home' und es werden die benötigten Templates und Controller geladen.

Ein Template mit Controller sieht in Angular wie folgt aus:

HTML: .

```
1           <div ng-controller="mainController as myController">
2                   <h1>{{myController.template}}</h1>
3           </div>
```

Angular.js .

```
1           app.controller('mainController', function(){
2                   this.template = "Hier steht der Templatetext!"
3           })
```

Die Ausgabe sieht dann so ähnlich aus: .

```
1           Hier steht der Templatetext!
```

Im Controller kann man nun Funktionen und Zuweisungen schreiben mit denen man im HTML Code arbeiten kann. Zudem kann man im Funktionskopf

Dependencies bestimmen, wie z.B. andere Services oder den $scope um das 'this' weglassen zu können.

Das Backend wird mit PHP realisiert. Dabei stehen mehrere PHP Scripts mit eingeschränkten Funktionsumfang zur Verfügung um entweder aus der SQlite Database zu lesen, oder Zeilen zu updaten/einfügen/löschen. Wie so ein Datenbank-Zugriff in einem PHP Script aussieht wird im folgenden Codeausschnitt dargestellt:

.

```
1   <?php
2       try {
3           // Use database file "database.sqlite"
4           $db = new PDO('sqlite:../../sqlite/database.
                sqlite');
5
6           // Throw exceptions on error
7           $db->setAttribute(PDO::ATTR_ERRMODE, PDO::
                ERRMODE_EXCEPTION);
8
9           $name = $_POST["request"];  //must be Username
10
11          //Vorzeitiges erstellen der Query um injection-
                attacks zu vermeiden
12          $query = "SELECT * FROM Users WHERE Username =
                '".$name."'";
13
14              $results = $db->query($query);
15
16              $rows = $results->fetchAll(PDO::FETCH_ASSOC)
                    ;
17              $data = array();
18
19              foreach ($rows as $rowKey => $row) {
20                  $data[] = $row;
21                  break;  //Break after 1 result.
                        Should do that automatically,
                        since Username is UNIQUE
22              }
23              echo json_encode($data);        //Convert
                    the data to a JSON String since we use
                    JSON as Object format
24          $db = NULL;         // Close database
25
26      } catch(PDOException $e)        {
27          echo "ERROR you noob :P -> ";
28          echo $e->getMessage();
29          echo $e->getTraceAsString();
30      }
31  ?>
```

Diese Methode des seperierten Statement-Erstellen und folgenden Ausfüren bietet relativ guten Schutz vor Injektions-Angriffen, bei denen man die Variablen der Abfrage ändert und somit den Query zwecksentfremdet und an Daten kommt, die man schützen wollte. Nun wird ein Service geschrieben, der dem Backend die Request-Daten schickt, die Antwort abfängt und die-

se in ein Object erstellt. Das wird mit Angular-Services realisiert, die im Folgenden beschrieben werden: .

```
1         angular.module('app.services.userDataService', [])
2
3
4                 .factory("userDataService", function($http){
5
6                 //Dummy Daten, falls es zu einem Fehler kommt um
                      undefinierte Zust\''ande zu vermeiden
7                 var userData =  {
8                 "firstname": "Unknown",
9                 "lastname": "Intruder",
10                "username": "(no name)",
11                "password": "0000",
12                "mail": "nomail@me.com",
13                "leaderboard": true,
14                "balance": -6.66,
15                        "messages": [
16                                {"Sender":"System","Type":"alert-
                                    danger","Title":"Hey, intruder
                                    ","Message":"I'm watching you
                                    ."},
17                        ],
18                        "history": [
19                                {"history_id":"0", "title": "No
                                    recent purchases", "price":"","
                                    date": ""},
20
21                        ],
22                        "favorites": [
23                        {"favorite_id":"0", "title": "Kaffee L", "
                              price":"1.50"},
24                        {"favorite_id":"1", "title": "Cappuchino", "
                              price":"1.20"},
25                        {"favorite_id":"2", "title": "Snickers", "
                              price":"0.50"},
26                        {"favorite_id":"3", "title": "Kaffee M", "
                              price":"1"},
27                        {"favorite_id":"4", "title": "Macchiato", "
                              price":"1.50"}
28                        ]
29                };
30
31                //Funktionen zum getten und setten dieser Userdaten
32                function getUserData() {
33                        return userData;
34                }
35                function setUserData(newData) {
36                        userData = newData;
37                }
38                return {
39                getUserData: getUserData,
40                setUserData: setUserData,
41
42                //Diese Funktion wird aufgerufen um Daten vom
                      Backend anzufordern
43                getUserDataSync: function(postData){
44
45                                                  //Hier wird der Ajax
                                                        Befehl
                                                    vorbereitet
```

```
46                 return  $.ajax({
47                 type: "POST",
48                 async: false,
49                 url: "/apis/users/getUserData.php",
50                 data: postData,
51                 success: function(response){
52                         console.log("Successful service request:
                               getUserDataSync");
53                         return response.data;   //Daten werden zum
                               ausf\''uhrenden Controller zur\''
                               uckgegeben
54                                                                 },
55                         //Fehlerbericht im Falle eines Backend-
                               Errors
56                         error: function(jqXHR, status, errors){
57                         console.log("Error service request");
58                         console.log(status);
59                                                                 }
60                                                         });
61                                 },
62
63                         }
64                 });
```

In den Servicefunktionen zum Datenfetching aus dem Backend werden im Falle eines Fehlers ein ausreichender Fehlerbericht in die Console ausgegeben. Dieser reicht meißtens aus um die Fehlerquelle zu lokalisieren.

Damit sich der User nicht jedes mal anmelden muss, wenn dieser die Webapp neu lädt, oder die Seite kurzzeitig verlässt, arbeiten wir mit Sessions, in denen wir die wichtigsten Eckdaten speichern, die der Server braucht um den Client einem gespeicherten User zuzuordnen. Das login-Script und Erstellen der Session-Cookies wird im Folgendem gezeigt: (login.php) .

```
1          <?php
2                  try {
3                          session_start();
4                      // Use database file "database.sqlite"
5                      $db = new PDO('sqlite:../../sqlite/database.
                           sqlite');
6                      // Throw exceptions on error
7                      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::
                           ERRMODE_EXCEPTION);
8
9                      if (isset($_POST["username"])){$username =
                           $_POST["username"];}else{$username = "";}
                                   //must be Username
10                     if (isset($_POST["password"])){$password =
                           $_POST["password"];}else{$password = "";}
                                   //must be Password
11
12                     $query = "SELECT * FROM Users WHERE Username =
                           '".$username."' AND Password = '".$password
                           ."'";
13                     $results = $db->query($query);
14                     $rows = $results->fetchAll(PDO::FETCH_ASSOC)
                           ;
15                     $data = array();
16                     foreach ($rows as $rowKey => $row) {
```

```
17                                  array_push($data,$row);
18
19                          }
20                          if (!empty($data)){
21                                  echo 'true';//json_encode($data);
22
23                                  $_SESSION['username'] = $data[0]['
                                        Username'];
24                                  $_SESSION['user_ID'] = $data[0]['
                                        User_ID'];
25                                  $_SESSION['rights'] = $data[0]['
                                        Rights'];
26                                  //$_SESSION['password'] = $data[0]['
                                        Password'];
27
28                          }
29                          else{
30                                  echo 'nodata';
31                          }
32                          // http_response_code(404);
33                      $db = NULL;         // Close database
34
35                  } catch(PDOException $e)        {
36                      echo "ERROR you noob :P -> ";
37                      echo $e->getMessage();
38                      echo $e->getTraceAsString();
39                  }
40          ?>
```

(retrieveData.php) .

```
1           <?php
2                   try {
3                           session_start();
4
5                           $data = array();
6                           // foreach ($rows as $rowKey => $row) {
7                           //      array_push($data,$row);
8                           // }
9                           foreach ($_SESSION as $key=>$value) {
10                                  //array_push($data,$value);
11                                  $data[$key] = $value;
12                          }
13                          if (!empty($data)){
14                                  echo json_encode($data);
15                          }
16                          else{
17                                  echo "nodata";
18                          }
19
20                  } catch(PDOException $e)        {
21                      echo "ERROR you noob :P -> ";
22                      echo $e->getMessage();
23                      echo $e->getTraceAsString();
24                  }
25          ?>
```

Falls keine Daten oder falsche Daten eingegeben werden, gibt das Backend ein 'nodata' Object zurück, welches im Frontend behandelt wird. Andernfalls bekommt das Frontend die gespeicherten Userinformationen und kann damit

die fehlenden Daten aus der DB nachreichen. Um so eine Session manuell zu löschen, gibt es auch ein logout.php Script: .

```
1   <?php
2           try {
3                   session_start();
4
5                   session_destroy();
6
7                   echo true;
8
9
10
11
12
13
14          } catch(PDOException $e)        {
15              echo "ERROR you noob :P -> ";
16              echo $e->getMessage();
17              echo $e->getTraceAsString();
18          }
19  ?>
```

5 Fazit des Tally Projekts

Die vorgegebenen Ziele konnten alle erfüllt werden. Der Raspberry besitzt alle erforderliche Hardware und Software um einen Webserver darauf laufen lassen zu können und um das Programm darauf ausführen zu können. Der Touchscreen und der Barcodescanner arbeiten auch ohne Störungen.

Auch die selbstgeschriebene Software Tally arbeitet einwandfrei und der Einkauf von Produkten funktioniert. Der Zugriff auf die Datenbank erfolgt ebenfalls, sodass die dort gespeicherten Informationen über die Website eingesehen und geändert werden können.

5.1 Probleme und Schwierigkeiten

5.1.1 Raspberry Pi 2

Probleme gab es zum Teil bei der Einrichtung von Backups auf dem Raspberry, die aber durch eine weit verbreitete Methode, rsync, behoben werden konnten. Wie zunächst gedacht konnte allerdings leider nicht das Barcodescannermodul verwendet werden, da die Spannung am Raspberry zu hoch ist, weshalb wir auf einen USB Barcodescanner umgestiegen sind.

5.1.2 Tally

5.1.3 Website

Ein Problem ist, dass man alle Daten aktuell halten muss, die sich bei jeder Benutzereingabe ändern können. Um dies zu gewährleisten, müssen Variablen unter *watchgestelltwerdenoderim*rootScope abgespeichert werden, was Ressourcen kostet, die im kleinen Umfang jedoch umbeträchtlich sind.

Zudem muss man seht genau auf Timing achten, da es keine Refreshes oder sonstige Events gibt, die Änderungen von Daten Triggern. Ajax bedeutet auch, dass Daten evtl. erst dann fertig geladen werden wenn das Template schon eingesetzt wurde. Das kann aber mit angulars eigenen Tags und Attributen umgangen werden, indem man z.B. anstatt 'src=' die Angular-Variante 'ng-src=' benutzt, welche darauf ausgelegt ist, die Daten später zu erhalten.

5.2 Verbesserungen und mögliche Änderungen in der Zukunft

5.2.1 Raspberry Pi 2

Als Verbesserung für den Raspberry könnte man einen größeren Touchscreen verwenden, da in Zukunft noch mehr Hardware für den Raspberry erscheinen wird. Auch wäre der Umstieg auf den neuen Raspberry Pi 2 B+ denkbar der durch seine bessere Hardware schneller arbeitet. Desweiteren könnte man ein Barcodescannermodul einbauen sodass der Raspberry noch handlicher wird.

5.2.2 Tally

5.2.3 Website

In Zukunft muss der Code deutlich sauberer gemacht werden und auskommentierter Code, der für Testzwecke drin geblieben ist, muss entfernt werden. Außerdem können Verbesserung zum Thema Sicherheit gemacht werden. Dazu gibt es noch etliche Angular-Funktionen und Angular-PlugIns. Letztendlich kann mit dem BootstrapUI-Plugin Bootstrap besser mit Angular verbunden werden, jedoch fehlte die Zeit um den Code zum wiederholten mal umzustrukturieren.

Literatur

[1] *C-Berry Touch von admatec.* http://admatec.de/pdfs/C-Berry_Touch.pdf. – Eingesehen am 14.07.2015

[10] *send mail.* http://www.gtkdb.de/index_36_2296.html. – Eingesehen am 02.07.2015

[11] *send mail with attachment.* http://www.gtkdb.de/index_36_2678.html. – Eingesehen am 02.07.2015

[12] *X11 (LXDE) Programme automatisch starten.* http://www.forum-raspberrypi.de/Thread-tutorial-automatisches-starten-von-scripte-programme-autostart pid=29112#pid29112. – Eingesehen am 14.07.2015

[13] *2015 The Qt Company.* http://www.qt.io/. – Eingesehen am 14.07.2015

[14] *Qt C++ tutorial by Brian.* https://www.youtube.com/watch?v=6KtOzh0StTc&list=PL3C390B298EDD3F86. – Eingesehen am 14.07.2015

[15] *QT C++ GUI Tutorial.* https://www.youtube.com/watch?v=8opfd5aYkq8&list=PLS1QulWo1RIZjrD_OLju84cUaUlLRe5jQ&index=1. – Eingesehen am 14.07.2015

[2] *Raspberry Remote Desktop.* http://www.xrdp.org/. – Eingesehen am 14.07.2015

[3] *PHP5.* http://php.net/manual/de/install.unix.debian.php. – Eingesehen am 05.07.2015

[4] *Qt Creator.* https://wiki.qt.io/Apt-get_Qt4_on_the_Raspberry_Pi. – Eingesehen am 08.07.2015

[5] *C-Berry Touch.* http://www.forum-raspberrypi.de/Thread-test-c-berry-3-5-tft-display. – Eingesehen am 08.07.2015

[6] *7 Zoll Pollin Touchscreen.* http://www.pollin.de/shop/dt/NTMwOTc4OTk-/Bauelemente_Bauteile/Aktive_Bauelemente/Displays/7_17_78_cm_Display_Set_mit_Touchscreen_LS_7T_HDMI_DVI_VGA_CVBS.html. – Eingesehen am 14.07.2015

[7] *4DPi-35 Touchscreen.* http://www.4dsystems.com.au/productpages/4DPi-35/downloads/4DPi-35_datasheet_R_1_3.pdf. – Eingesehen am 01.07.2015

[8] *Edimax Wlan Stick*. http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un. – Eingesehen am 14.07.2015

[9] *Bootlogo*. http://www.edv-huber.com/index.php/problemloesungen/15-custom-splash-screen-for-raspberry-pi-raspbian. – Eingesehen am 10.07.2015