

Tally

digitale Strichliste am Raspberry Pi

Nicolai Tegtmeier
Dominik Scheffler
Philip Frieling
Sebastian Reinke

23.06.2015



Abbildung 1: Tally Logo

Inhaltsverzeichnis

1 Projektvorfeld	3
1.1 Kaffeestrichliste bisher	3
2 Rahmenbedingungen	3
2.1 Aufgabe - die neue Kaffeestrichliste	3
3 Ziele des Projekts	3
3.1 Zielbestimmung	4
3.1.1 Der Benutzer Account	4
3.1.2 Das Programm	4
3.1.3 Der Administrator Account	4
3.2 Produkteinsatz	4
3.2.1 Anwendungsbereiche	4
3.2.2 Zielgruppen	4
3.2.3 Betriebsbedingungen	4
3.3 Produktumgebung	5
3.3.1 Software	5
3.3.2 Hardware	5
3.3.3 Orgware	5
3.4 Produktfunktion	5
3.4.1 Benutzerfunktion	5
3.5 Programmfunktion	6
3.6 Serverfunktion	6
3.6.1 Datenbank	6
3.7 Benutzerschnittstelle	6

4 Meilensteine des Projektes	7
4.1 Raspberry Pi 2 Model B - Die Hardware mit der passenden Software	7
4.1.1 Betriebssystem und Verbindung zu anderen Rechnern	7
4.1.2 Der Webserver	8
4.1.3 Qt Creator	8
4.1.4 Die passende Datenbank	10
4.1.5 Der Touchscreen	10
4.1.6 Produkte scannen	12
4.1.7 Wlan für den Raspberry	12
4.1.8 angepasster Bootscreen	13
4.1.9 Backups einrichten und versenden per Mail	15
4.1.10 Tally autostarten	20
4.2 QT Creator - Das Tally Programm	23
4.2.1 Die Benutzeroberfläche	23
4.2.2 Datenbank und Tally	23
4.2.3 Aufbau des Programmes	25
5 Auflistungen und Aufzählungen	35
6 Tabellen, Grafiken und Gleitobjekte	36
6.1 Grafiken	36
6.2 Tabellen	36
6.3 Bewegliche Objekte	36

1 Projektvorfeld

1.1 Kaffeestrichliste bisher

In kleineren Unternehmen und Arbeitsgruppen gibt es oft einen zentralen Kaffee/ Getränke -automaten und Snacks, an denen sich jeder Mitarbeiter bedienen kann. Um die Getränke und Snacks kaufen zu können wird meistens eine Strichliste auf Papier, meist 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 das 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 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 mithilfe 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. Des Weiteren soll sie täglich vierundzwanzig Stunden laufen.

3.3 Produktumgebung

Das Produkt kann unabhängig an jedem Ort betrieben werden, solange 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 zuweisen. 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 Produkt einscannen, welches im Warenkorb erscheint. Des Weiteren 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 Programmfunction

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 Steckleiste, ein HDMI Anschluss, ein Ethernet Port und ein Micro SD Kartenslot am Raspberry.

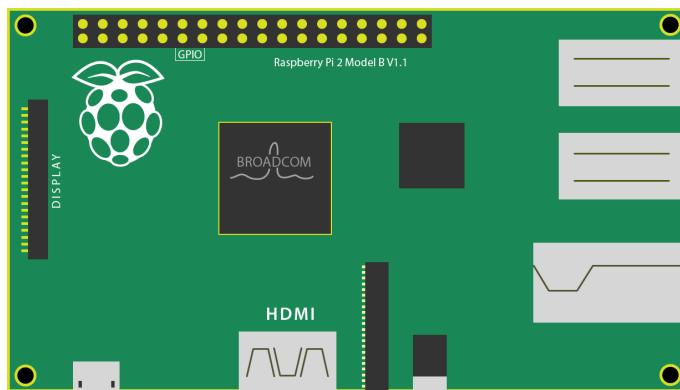


Abbildung 2: Der Raspberry Pi 2

Ausser dem Raspberry Pi 2 war zunächst ein 3,5 Zoll Touchscreen der Firma admatec [10] mit dem Namen C-Berry Touch vorhanden, das mittels eines Adapterboards an der GPIO Stiftleiste des Raspberry angesetzt 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 das 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. [1].

```
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 nächster Schritt 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 Websites zu gewährleisten. [2]

PHP ist eine Open Source-Skriptsprache die speziell für die Webprogrammierung geeignet ist. Wenn eine Anfrage an die Website gestellt wird, 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 programmiert und auf Richtigkeit überprüft. Erst dann wird die Version auf GitHub zur Verfügung gestellt und auf dem Raspberry lauffähig gemacht.

Damit das Programm auf dem Raspberry lauffähig gemacht werden kann muss der Qt Creator samt Qt4 Umgebung installiert werden. Dieser kann dann das gespeicherte unkompliierte Programm öffnen und passend für den Raspberry kompilieren.

Das Programm auf einem anderen Computer zu kompilieren fällt weg, 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 .

```
1| sudo apt-get install qtcreator
2| sudo apt-get install qt4-dev-tools
3| sudo apt-get install gcc
4| sudo apt-get install xterm
5| sudo apt-get install git-core
6| 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 .

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

beim Debugger .

```
1| /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]

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. .

```
1| apt-get install sqlite3  
2| 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

Als erstes wurde der Treiber für den bereitgestellte C-Berry Touchscreen installiert [5] .

```
1| wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.36.tar.gz  
2| tar zxvf bcm2835-1.36.tar.gz  
3| cd bcm2835-1.36  
4| ./configure  
5| make  
6| sudo make check  
7| sudo make install
```

und im Anschluss die Beispielsoftware zum anzeigen von Bildern installiert .

```
1| wget http://admatec.de/sites/default/files/downloads/C-Berry.tar.gz  
2| tar zxvf C-Berry.tar.gz  
3| cd C-Berry/SW/tft_test  
4| make  
5| sudo ./tft_test
```

Nach ersten Tests mit dem C-Berry Touchscreen fiel auf das 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 mehr 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 Raspberry und einer höheren Bildwiederholungsrate musste gesucht werden. Dabei kam die Idee auf, ein 7 Zoll resistives Touchscreen von Pollin zu verwenden das 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 Bildwiederholungsrate ermöglicht 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.[6] Der Touchscreen besitzt eine Auflösung von 480x320



Abbildung 3: Der 4 DPi-35 Touchscreen

Pixeln und durch die High Speed 48Mhz SPI Verbindung werden hohe und konstante Bildwiederholungsraten ermöglicht.

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 ein Barcodescannermodul nur mit ca. drei Volt betrieben werden kann, der Raspberry aber ca. fünf Volt an den GPIO's zur Verfügung stellt, wurde überlegt eine Webcam an den Raspberry anzuschliessen und mittels einer Software das Bild auf Barcodes zu untersuchen. Die Wahl fiel allerdings auf das Scannen mittels USB Barcodescanner, da dies eine einfacherere Handhabung mit 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, wurde der Edimax WLAN USB Stick am Raspberry angeschlossen. Der Edimax WLAN Stick wird automatisch von Raspbian erkannt da schon alle erforderlichen Pakete und Treiber installiert sind, weshalb dieser WLAN Stick empfohlen wird falls man WLAN am Raspberry benutzen möchte. Zunächst wurde in der automatische 'Power Saving' Modus abgeschaltet.

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

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

mit folgendem Inhalt .

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

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

```
1| sudo nano /etc/network/interfaces
```

um folgenden Inhalt ergänzt .

```
1| auto lo
2| iface lo inet loopback
3| iface eth0 inet dhcp
4|
5| auto wlan0
6| allow-hotplug wlan0
7| iface wlan0 inet dhcp
8| wpa-ap-scan 1
9| wpa-scan-ssid 1
10| wpa-ssid "WLAN-NAME"
11| wpa-psk "WLAN-PASSWORT"
```

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

```
1| 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 un 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 das während des hochfahrens angezeigt werden kann. [7]

```
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 .

```
1#!/bin/sh
2### BEGIN INIT INFO
3# Provides: asplashscreen
4# Required-Start:
5# Required-Stop:
6# Should-Start:
7# Default-Start: S
8# Default-Stop:
9# Short-Description: Show custom splashscreen
10# Description: Show custom splashscreen
11### END INIT INFO
12
13do_start () {
14    /usr/bin/fbi -T 1 -noverbose -a /etc/splash.png
15    exit 0
16}
17
18case "$1" in
19    start|")
20        do_start
21        ;;
22    restart|reload|force-reload)
23        echo "Error: argument '$1' not supported" >&2
24        exit 3
25        ;;
26    stop)
27        # No-op
28        ;;
29    status)
30        exit 0
31        ;;
32        ;;
33    *)
34        echo "Usage: asplashscreen [start|stop]" >&2
35        exit 3
36        ;;
37esac
38
39:
```

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

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

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

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

Nun wird während des Bootvorgangs das Tally Logo angezeigt.

4.1.9 Backups einrichten und versenden per Mail

Als mögliche Backupvarianten standen ein komplettes Backup der Speicher-karte 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:

```
1| sudo apt-get install rsync
```

Rsync ist ein Programm das 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 automatisch und täglich erstellt werden, wird ein cronjob hinzugefügt. Cronjobs, oder auch Cron-Daemon, ist ein Dienst um Skripte oder Programme zu einer bestimmten Zeit starten zu können. Zunächst wird die Konfigurationsdatei von cronjob geöffnet .

```
1| sudo crontab -e
```

Hier wird folgende Zeile eingefügt .

```
1| 15 22 * * * rsync -av --delete /var/www/sqlite/database.sqlite /  
          home/pi/Tally/sicherung
```

Dadurch wird das Programm jeden Tag um 22:15Uhr aufgerufen und die Datenbank 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 jeden Tag 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..

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

SendEmail hat den Vorteil das kein Mailserver auf dem Raspberry installiert werden muss und der Raspberry als ein eigener 'sende Client' erkannt wird. 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 [8] .

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

Das Skript sieht wie folgt aus .

```
1 #####  
2  
3             Sending E-Mail notification with sendEmail  
4  
5 Creation:      15.08.2013  
6 Last Update: 20.10.2013  
7  
8 Copyright (c) 2013 by Georg Kainzbauer <http://www.gtkdb.de>  
9  
10 This program is free software; you can redistribute it and/or  
11     modify  
12 it under the terms of the GNU General Public License as published  
13     by  
14 the Free Software Foundation; either version 2 of the License, or  
15 (at your option) any later version.  
16 #####  
17  
18 #!/bin/bash  
19  
20 # Sender of the mail  
21 SENDER="absender@domain.de"  
22  
23 # Recipient of the mail  
24 RECIPIENT="empfaenger@domain.de"  
25  
26 # SMTP server  
27 SMTPSERVER="smtp.domain.de"  
28  
29 # User name on the SMTP server  
30 SMTPUSERNAME="MeinMailAccount"  
31  
32 # Password on the SMTP server  
33 SMTPPASSWORD="MeinMailPasswort"  
34  
35 # Enable TLS for the SMTP connection  
36 USETLS=1  
37 #####  
38 # NORMALLY THERE IS NO NEED TO CHANGE ANYTHING BELOW THIS COMMENT #  
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 # Generate the options list for sendEmail  
57 OPTIONS=""
```

```

58 if [ -n "${SMTPSERVER}" ]; then
59   OPTIONS="${OPTIONS} -s ${SMTPSERVER}"
60 fi
61
62 if [ -n "${SMTPUSERNAME}" ]; then
63   OPTIONS="${OPTIONS} -xu ${SMTPUSERNAME}"
64 fi
65
66 if [ -n "${SMTTPASSWORD}" ]; then
67   OPTIONS="${OPTIONS} -xp ${SMTTPASSWORD}"
68 fi
69
70 if [ -n "${USETLS}" ]; then
71   if [ ${USETLS} == 1 ]; then
72     OPTIONS="${OPTIONS} -o tls=yes"
73   else
74     OPTIONS="${OPTIONS} -o tls=no"
75   fi
76 fi
77
78 # Send the mail with sendEmail
79 sendEmail -f ${SENDER} -t ${RECIPIENT} -u "${SUBJECT}" -m "${BODY}"
80           ${OPTIONS}
81
82 exit 0

```

Zu editieren sind folgende Punkte: .

```

1 # Sender of the mail
2 SENDER="absender@domain.de"
3
4 # Recipient of the mail
5 RECIPIENT="empfaenger@domain.de"
6
7 # SMTP server
8 SMTPSERVER="smtp.domain.de"
9
10 # User name on the SMTP server
11 SMTPUSERNAME="MeinMailAccount"
12
13 # Password on the SMTP server
14 SMTTPASSWORD="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 Benutzernamen und Passwort angegeben werden. Danach muss das Skript ausführbar gemacht werden .

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

Im Anschluss kann die erste Email versendet werden .

```
1| 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 zweite Anführungszeichen den Inhalt der Email. Hierbei kam es jedoch zu folgendem Fehler .

```
1| 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 enthält. Folgende Datei muss aufgerufen werden .

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

und in Zeile 1490 muss folgender Inhalt .

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

durch .

```
1| 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 .

```
1| mailnotify.sh "Test" "Dies ist eine Testnachricht."  
2| 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. [9] .

```
1| # Use third argument as attachment  
2| if [ -n "$3" ]; then  
3|   ATTACHMENT="$3"  
4| else  
5|   # No attachment specified  
6|   ATTACHMENT=""  
7| fi
```

Somit sieht das Skript wie folgt aus [8] .

```
1 #!/bin/bash
2
3 # Sender of the mail
4 SENDER="absender@domain.de"
5
6 # Recipient of the mail
7 RECIPIENT="empfaenger@domain.de"
8
9 # SMTP server
10 SMTPSERVER="smtp.domain.de"
11
12 # User name on the SMTP server
13 SMTPUSERNAME="MeinMailAccount"
14
15 # Password on the SMTP server
16 SMTPPASSWORD="MeinMailPasswort"
17
18 # Enable TLS for the SMTP connection
19 USETLS=1
20
21 ######
22 # NORMALLY THERE IS NO NEED TO CHANGE ANYTHING BELOW THIS COMMENT #
23 #####
24
25 # Use first argument as mail subject
26 if [ -n "$1" ]; then
27     SUBJECT="$1"
28 else
29     # No subject specified
30     SUBJECT=""
31 fi
32
33 # Use second argument as mail body
34 if [ -n "$2" ]; then
35     BODY="$2"
36 else
37     # No mail body specified
38     BODY=""
39 fi
40
41 # Use third argument as attachment
42 if [ -n "$3" ]; then
43     ATTACHMENT="$3"
44 else
45     # No attachment specified
46     ATTACHMENT=""
47 fi
48
49 # Generate the options list for sendEmail
50 OPTIONS=""
51
52 if [ -n "${SMTPSERVER}" ]; then
53     OPTIONS="${OPTIONS} -s ${SMTPSERVER}"
54 fi
55
56 if [ -n "${SMTPUSERNAME}" ]; then
57     OPTIONS="${OPTIONS} -xu ${SMTPUSERNAME}"
58 fi
59
60 if [ -n "${SMTPPASSWORD}" ]; then
61     OPTIONS="${OPTIONS} -xp ${SMTPPASSWORD}"
```

```

62 fi
63
64 if [ -n "${USETLS}" ]; then
65   if [ ${USETLS} == 1 ]; then
66     OPTIONS="${OPTIONS} -o tls=yes"
67   else
68     OPTIONS="${OPTIONS} -o tls=no"
69   fi
70 fi
71
72 if [ -n "${ATTACHMENT}" ]; then
73   OPTIONS="${OPTIONS} -a ${ATTACHMENT}"
74 fi
75
76 # Send the mail with sendEmail
77 sendEmail -f ${SENDER} -t ${RECIPIENT} -u "${SUBJECT}" -m "${BODY}"
    ${OPTIONS}
78
79 exit 0

```

Im Befehl zum senden wird nun ein drittes Argument hinzugefügt das den Pfad mit der Datei die zu versenden ist enthält..

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

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

```
1| sudo crontab -e
```

hinzugefügt wird .

```
1| 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 23:45Uhr 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 das das Programm startet, welches dann in einem cronjob nach jedem hochfahren ausgeführt wird..

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

Mit folgendem Inhalt .

```
1#!/bin/sh
2### BEGIN INIT INFO
3# Provides: NAME DES PROGRAMMES
4# Required-Start: $syslog
5# Required-Stop: $syslog
6# Default-Start: 2 3 4 5
7# Default-Stop: 0 1 6
8# Short-Description:
9# Description:
10### END INIT INFO
11
12case "$1" in
13    start)
14        echo "NAME DES PROGRAMMES wird gestartet"
15        # Starte Programm
16        /pfad/des/programmes
17        ;;
18    stop)
19        echo "NAME DES PROGRAMMES wird beendet"
20        # Beende Programm
21        killall noip2
22        ;;
23    *)
24        echo "Benutzt: /pfad/des/programmes {start|stop}"
25        exit 1
26        ;;
27esac
28
29exit 0
```

Danach wird das Skript ausführbar gemacht .

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

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

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

und wieder gestoppt .

```
1| 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 .

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

Als nächstes soll das Skript durch einen cronjob jedes mal beim starten des Raspberries ausgeführt .

```
1| sudo crontab -e
```

Dazu wurde folgende Zeile eingefügt .

```
1| @reboot \bin\bash .....
```

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 der cronjob das Skript und somit das Programm.

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

```
1| /etc/xdg/autostart/
```

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

```
1| [Desktop Entry]
2| Name=NamedesProgramm
3| Comment=
4| Exec=NamedesProgramm -forever -rfbport 5900 -rfbauth ~/.vnc/x11vnc.
   pass -o ~/.vnc/x11vnc.log -display :0
5| Terminal=false
6| Type=Application
```

Diese Datei wieder ausführbar machen .

```
1| 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. Qt ist eine Klassenbibliothek in C++, vereinfacht die Programmierung für grafische 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. 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 bürge eine große Hürde. Das Problem mit der Synchronität des Shoppingcart über den gesamten Einkauf.

Jedes Mal wenn der Zustand 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 Shoppingcart erstellt nun vor jedem Löschen eine Liste mit seinen Items, welche dem neu erzeugten Shoppingcart 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 hinzu 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 folgenden 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 den 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. ->(MEALYAUTOMATBILD)<- 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
2      else reliable
3  if (socket.waitForConnected()) {
4      ip = socket.localAddress().toString();
5  } else {
6      ip = socket.errorString();
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.

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.

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 aktualisiert 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.

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 aktualisiert 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.

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 aktualisiert 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.

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 aktualisiert und man wird in den afterbuyscreen geschickt nach einigen Sekunden dann wieder in den Login-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 aktualisiert 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.

mainwindows.cpp:

Wie bereits erwähnt haben wir die Oberfläche in zwei Teile unterteilt, in welche dann Widget hinzugefügt und entfernt 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 Betriebssystem, 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 nutzen 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 wieder 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 angehängen. 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 eingelesenen 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, welche über die Datenbank eingefordert 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 Alphabets. 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 bedeutet 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 Eingebenen Codes sind. Realisiert haben wir dies, indem von jedem Benutzer der `T9_Code` in der Datenbank gespeichert ist und nun mit Hilfe 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 den Passwort-Screen. Außerdem wird bei diesem Befehl, die `User_Id` gespeichert, womit andere Funktionen wissen welcher Benutzer 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  
2|     != '1' \&\& database.checkUserLoginCount(userId)){ //  
3|     Passwort wurde korrekt eingegeben und der User ist  
4|     noch geblockt  
5|     database.updateLoginAttempt(userId,true);  
6|     Data.close();  
7|     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 geblockt. Dies geschieht in dem wir, bei den letzten misslungenen 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 gesperrt. Sobald man sich einmal richtig einloggt wird der Zeitstempel aktualisiert und der Zähler wieder 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 kurzzeitig gesperrt ist aufgrund zu häufiger Eingabe des falschen Passwortes, wird dieser auch geblockt. Dann erscheint anstelle 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 Getränkelisten oder die Süßigkeitenliste an. Welcher der

beiden Liste gezeigt wird, wird dem der Funktion über die setMainWindowPointer 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 explizit speichern und wenn wir ausgeben lassen was dort gespeichert ist, bekommen wir einen leeren String zurück.

Zusätzlich verändert sich die Farbe bei unterschiedlichen 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 Auswahlmöglichkeiten 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 kombiniert 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->  
4 |             data(6).toInt());  
5 |         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 synchronisieren 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, während 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 erscheint auf der linken Seite in einem QListWidget, während 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 bzw 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 Shoppingcart 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 Elemt

der Schlangen entfernt, wodurch das Element als nächstes ausgewertet werden kann. Falls kein Element mehr vorhanden ist, gibt die Funktion FALSE zurück.

Außerdem kann die Klasse auch in Datenbank schreiben. Einmal kann eine neue Spalte erzeugt werden durch

```
1 | -->query.exec(INSERT INTO Consum\_Index (User\_ID, Grocery\
2 | _ID, Count) VALUES(+userId+,+Grocery\_Id+,+count+));
   | Zuerst wird die Tabelle mit den Spalten, die gefüllt
   | werden sollen genannt Consum\_Index (User\_ID, Grocery\
   | _ID, Count) und dann werden die einzufügenden 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 muss 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 Raspberry 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 jedoch die Eigenschaft, mehrere Sekunden ein Screen zu zeigen auf dem der Benutzer erkennen kann, dass 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.

5 Auflistungen und Aufzählungen

Wir fassen zusammen, was wir bisher knnen:

- Die Schrift anpassen
- Den Text ausrichten. Da gab es folgende Mglichkeiten:
 - Blocksatz
 - linksndig
 - rechtsbndig
 - zentriert. Zentrierte Text ist vor allem gut um
 - * Text hervorzuheben
 - * Objekte, wie Tabellen und Grafiken zu zentrieren
- Umbrche erzwingen
- Sonderzeichen setzen

Bisher knnen wir noch nicht:

1. Tabellen erzeugen
2. Grafiken einbinden
3. mathematischen Text setzen. Dazu gehrt
 - (a) weitere mathematische Sonderzeichen, wie z.B.
 - i. griechische Buchstaben, etwa α , ζ , usw.
 - ii. spezielle Akzente, wie \vec{a} oder \ddot{x}
 - iii. echte Sonderzeichen, wie \oplus oder \perp
 - iv. alle Mglichen Klammern, wie $[x]$
 - v. und noch vieles mehr...
 - (b) spezielle mathematische Objekte, wie Matrizen
 - (c) automatische Nummerierung von Formeln
4. Referenzen und Bibliographie erzeugen
5. Prsentationsfolien erstellen

Offensichtlich kann man hier ziemlich tief schachteln. Ob das allerdings immer sinnvoll ist?

6 Tabellen, Grafiken und Gleitobjekte

6.1 Grafiken



Hier hatten wir Gtig kommt?

6.2 Tabellen

Name	gemessen von Dr. Tex		
	Alter (Jahre)	Gro (in cm)	Gewicht (in kg)
Andreas	7	120	25
	10	141	34
	14	163	50
Beate	6	110	22
	9	138	32
	13	156	46
Tina	8	132	30
	11	151	43
	15	174	51

6.3 Bewegliche Objekte

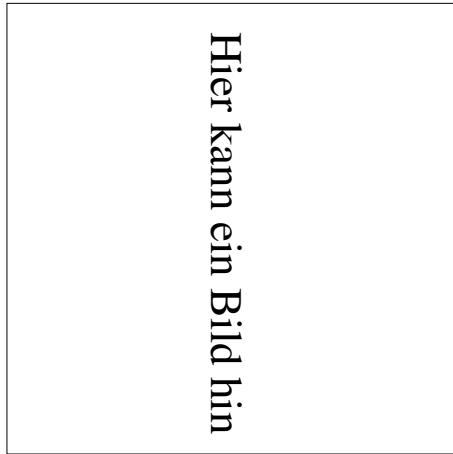


Abbildung 4: Dasselbe nochmal als Gleitobjekt.

Literatur

- [1] *Raspberry Remote Desktop*. <http://www.xrdp.org/>. – Eingesehen am 14.07.2015
- [10] *C-Berry Touch von admatec*. http://admatec.de/pdfs/C-Berry_Touch.pdf. – Eingesehen am 14.07.2015
- [2] *Webserver*. <http://www.forum-raspberrypi.de/Thread-tutorial-raspberry-pi-als-webserver-apache-2-installation>. – Eingesehen am 08.07.2015
- [3] *PHP5*. <http://wiki.ubuntuusers.de/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] *4DPi-35 Touchscreen*. http://www.4dsystems.com.au/productpages/4DPi-35/downloads/4DPi-35_datasheet_R_1_3.pdf. – Eingesehen am 01.07.2015
- [7] *Bootlogo*. <http://www.edv-huber.com/index.php/problemloesungen/15-custom-splash-screen-for-raspberry-pi-raspbian>. – Eingesehen am 10.07.2015

- [8] *send mail.* http://www.gtkdb.de/index_36_2296.html. – Eingesehen am 02.07.2015
- [9] *send mail with attachment.* http://www.gtkdb.de/index_36_2678.html. – Eingesehen am 02.07.2015