Beuth Hochschule für Technik Berlin

Spyhole

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences

Seminararbeit für die Lehrveranstaltung
Model Based Design bei Herr Flemming

Eingereicht am
24. Januar 2014

Eingereicht von

Matthias Hansert	s791744
Marcus Perkowski	s798936
Björn Hepner	s798148
Julian Zoellner	s798465
Niklas Knauer	s798163

Inhaltsverzeichnis

Kapitel 1

Einleitung

Kapitel 2

Raspberry Pi

Das Raspberry Pi ist ein kreditkartengroßer Einplatinencomputer von der *Raspberry Pi Foundation*[1] entwickelt wurde. Mithilfe seiner 700 MHz starken ARM-CPU kann er nahezu alles was auch normale Desktoprechner können.

Ziel der Entwicklung des Raspberry Pi ist das Interesse an dem Studium der Informatik und ähnliche Fachgebiete zu fordern, bzw. einen günstigen Computer zum Programmieren und Experimentieren anbieten zu können.

Durch die geringe Leistungsaufnahme, günstigen Preis und viele verschiedene Ausbau- und Personalisierungsmöglichkeiten können Raspberry Pi's für verschiedene Zwecke eingesetzt werden. Unter anderem sind beliebte Projekte für den Haushalt Wetterstationen, Radiosender, Media Center, NAS, etc.

[1] Stiftung und Wohltätigkeitsorganisation in Großbritannien

2.1 Hardware

2.1.1 Technische Spezifikationen

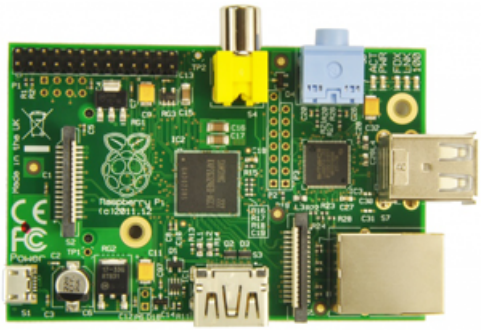

Abbildung 2.1: Raspberry Pi Model B

Testfall	Fehler
Größe	85,60 x 53,98 x 17 mm
Soc	Broadcom BCM2835
CPU	ARM1176JZF-S(700MHz)
GPU	Broadcom VideoCore IV
SDRAM	512MB
USB 2.0	2
Videoausgabe	HDMI & S-Video
Tonausgabe	3,5mm Klinkenstecker & HDMI
Speicher	SD(SDHC/SDXC)/MMC/SDIO Kartenleser bis 128GB
Netzwerk	10/100 MBit Ethernet
Schnittstellen	17 GPIO-Pins, SPI, 21C, UART
Stromversorgung	5V Micro-USB Anschluss

2

[2]http://www.raspberrypiguide.de; http://www.raspberrypi.org; 15.01.2014

2.1.2 Kamera

Abbildung 2.2: Raspberry Kamera

Testfall	Fehler
Größe	20 x 24 mm
Sensor	OmniVision OV5647 (5MP)
Auflösung	2592 x 1944
Video	1080p 30fps
Anschluss	15-Pin Flachbandkabel

2.2 Software und Einstellungen

2.2.1 Betriebssystem

Für das Raspberry Pi worden mehrere Open Source Betriebssysteme programmiert. Alle basieren auf verschiedene Linux Distributionen(u.a. OpenSUSE, Fedora, FreeBSD, Debian). Je nach Bedarf kann sich jeder Nutzer für eine Distribution entscheidend, eins des beliebtestens Raspbian, basierend auf Debian, was auch in diesem Projekt benutzt wird. Raspbian inkludiert alle Grundprogramme und -Utilities dass mit vielen verschiedenen Packages kompatibel sind. Dieser Betriebssystem hat sich im laufe der Zeit als sehr stabil bewiesen.

Es gibt auch die Möglichkeit Windows auf das Raspberry Pi zu installieren, obwohl Nutzer davon abgeraten werden. Das Windows Betriebssystem benötigt zu viele Ressourcen und läuft sehr langsam und instabil auf das Raspberry Pi.

Andere Distributionen sind Gebrauchorientiert aufgestellt, wie zum Beispiel für Media Center das beliebte XBMC (OpenELEC, Raspbmc, XBian). Weiterhin gibt es auch Androidsysteme die auf das Raspberry Pi portiert worden sind. [3]

[3] http://www.raspbian.org; 15.01.2014

2.2.2 LAMP Webserver

LAMP steht für Linux Apache MySQL PHP Webserver. Auf das Raspberry Pi wurde ein LAMP Server installiert um die Desktop/Mobil Applikation mit dem Pi zu verbinden und Informationen Austauschen. Durch die Einstellung eines DNS kann auf das Videostreams sowie auf die MySQL Datenbank zugegriffen werden. Damit *http* oder *ssh* Anfragen an das Server im Raspberry Pi ankommen, müssen Ports für die Verbindung zur Verfügung gestellt werden. Im diesen Fall muss jeweils ein Port für die SSH-Verbindung, den Videostream und die Datenbankzugriffe freigegeben werden.

Um das LAMP Webserver zu installieren werden folgende Kommandos mit Administratorrechte ausgeführt:

```
apt-get install apache2
apt-get install mysql-server
sudo apt-get install php5
sudo apt-get install php5-mysql
```

2.2.3 SSH Zugriff

Damit das Team gleichzeitig auf das Raspberry Pi arbeiten konnte wurde auf das System eine SSH-Verbindung eingestellt. So wurden auch Entwicklungskosten gespart, weil nicht jeder Teammitglied ein System benötigte.

Um das Raspberry Pi per Fernzugriff zu betreiben ist es nötig eine *Secure Shell* Verbindung aufzubauen. Durch die Verfügung einer SSH-Verbindung können Updates, Support und Wartungsarbeiten per Fernzugriff auf das System ausgeübt werden, dieses spart kosten für den Kunden sowie für den Support.

Als Sicherheitsmaßnahmen wurde der Standardport für SSH-Verbindungen (Port 22) auf ein anderen Port geändert und eine Public Key Authentifizierung implementiert. Um den Port für die SSH-Verbindung zu ändern, muss in der Datei */etc/ssh/sshd_config* der Wert *#Port* auf den gewünschten Wert gesetzt werden. Danach muss das Dienst neu gestartet werden und somit ist der neue Port eingestellt.

Die Public Key Authentifizierung erfolgt indem der Benutzer ein privaten und einen öffentlichen Schlüssel erstellt. Der private Schlüssel wird am Client gespeichert und der öffentlicher Schlüssel am Server. Mit dem Befehl

```
ssh-keygen -t dsa
```

werden die Schlüsseln generiert. Hier muss der Benutzer ein Passwort eingeben, mit dem sich er am Server anmelden wird. In der Regel heißen beider Schlüssel *id_das* und *id_das.pub*. Der private Schlüssel muss an das Client bekannt gemacht werden, indem man in der */ssh2/identification* Datei die Zeile

```
idkey id_dsa
```

bearbeitet oder einbaut. Der Inhalt des öffentlichen Schlüssels muss in der Datei */ssh/authorized_keys* hinzugefügt werden.

```
cat new_key.pub >> .ssh/authorized_keys
```

Jetzt ist der Client und der Server so konfiguriert, dass nur Rechner mit dem privaten Schlüssel Zugriff auf den Server haben über den konfigurierten Port. Eine SSH-Verbindung wird mit folgendermaßen aufgebaut:

```
ssh -p XXXX benutzer@serverAdresse
```

2.2.4 DNS

Um das Raspberry Pi über das lokale Netz erreichbar zu machen, wurde erstmal ein Webserver aufgebaut. Um jetzt das System überall über das Internet erreichbar ist, muss eine Domain reserviert werden. Somit kann ein Benutzer Zugriffe auf die Datenbank, sowie auf das Videostream aus jedem Rechner oder Android Mobilgerät weltweit ausüben. Der Dienst “no ip” bietet solche Dienstleistung kostenlos an. Für dieses Projekt wurde die Adresse *spyhole.no-ip.biz* eingestellt und durch die Freigabe und Weiterleitung von Ports können die programmierten Applikationen auf das Raspberry Pi zugreifen.

Da dieses Projekt im eine privaten Haushalt verwendet wird, besitzt in der Regel der Benutzer keine feste IP Adresse. Deswegen muss die IP Adresse hinter des DNS zyklisch geprüft werden. Der Dienst “no ip” stellt ein Programm zur Verfügung, das sich um die Überprüfung der Gültigkeit der IP Adresse kümmert. Das Programm muss dann mit den Kommandos

```
make
make install
```

installiert werden. Während der Installation wird von den Benutzer das Login, Passwort und die Aktualisierungszeit verlangt. Danach wird der Dienst mit

```
/usr/local/bin/noip2
```

gestartet. Der Dienst läuft bis das System runter gefahren wird. Daher ist es ratsam der Dienst im */etc/init.d* einzutragen.

2.3 OpenCV

Ist eine freie Programmbibliothek(unter BSD-Lizens) mit Algorithmen für Bildverarbeitung und maschinelles sehen. Es beinhaltet C/C++, Python und Java Interfaces und unterstützt Windows, Linux, Mac OS, iOS and Android. OpenCV wird sowohl im kommerziellen wie im privaten Bereich stark benutzt.

2.3.1 Gesichtserkennung

Unter die vielen Möglichkeiten das OpenCV anbietet, ist für dieses Projekt die Funktionen der Gesichtserkennung relevant.

hier kommt noch text

2.4 MJPG Streamer

hier kommt noch text

2.4.1 subsection

hier kommt noch text

Kapitel 3

Android App

3.1 Android

Das Betriebssystem ist komplett in der Sprache Java programmiert worden. Dabei handelt es sich um ein Open Source Betriebssystem, welches von der Firma Google entwickelt worden ist. Kern des Betriebssystems ist ein angepasster Linux-Kernel 2.6.

3.1.1 Struktur und Aufbau der App

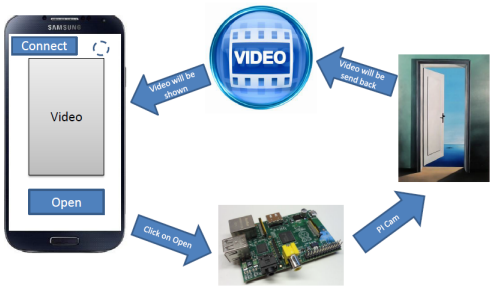

Abbildung 3.1: Prozess der App

Sobald die Control-View geladen ist verbindet sich das mobile Endgerät mit dem Raspberry Pi. Nachdem die Verbindung aufgebaut wurde, kann der Benutzer der App die Schaltfläche `Open` betätigen. Nach der erfolgreicher Verbindung zum Server sendet dieser einen kontinuierlichen Live-Stream an das Endgerät. Wenn der Benutzter nun die Schaltfläche `Open` betätigt, wird ein Befehl zum Öffnen der Tür gesendet und ein GPIO angesteuert, der den Türöffner betätigt, um die Tür zu öffnen.

3.1.2 Anmelden des Users

Um zu verhindern, dass nicht jede Person auf den Stream zugreifen kann, sollte aus Sicherheitsaspekten ein Login implementiert werden. Da aber unerwartete technische Probleme auftraten und diese in der Zeit nicht gelöst werden konnte wurde dieser Aspekt, erst einmal beiseite gestellt. Jedoch wird im Folgenden auf die Vorgehensweise eingegangen, welche Technologien verwendet worden sind und wie der Login- und Registrierungsprozess ablaufen soll beschrieben.

Registrierung

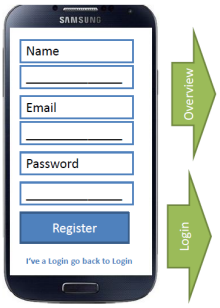

Abbildung 3.2: Registrierung

Um sich überhaupt einloggen zu können muss sich der Benutzer beim aller ersten Mal mit seinen Kontaktdaten registrieren. Dabei muss der Nutzer seine

- E-Mail
- Nutzername
- Passwort

eingeben. Das Framework Android-SDK bietet es dem Nutzer an die grafische Oberfläche von der eigentlich Logik zu trennen. Die grafischen Oberflächen werden als Views betitelt. Die Views werden mithilfe von XML gestaltet.

```
<Button
       android:id="@+id/btnLinkToLoginScreen"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="40dip"
       android:background="@null"
       android:text="Already␣registred.␣Log␣Me␣In!"
       android:textColor="#21dbd4"
       android:textStyle="bold" />
```

Listing 3.1: Android - Button erstellen

In dem obigen Codebeispiel wird ein Button in der View erzeugt. Dabei wird dieser mit einer sog. ID versehen, um den Button über diese ID aus dem Quellcode ansprechen zu können. Mithilfe von `android:layout_width` und `android:layout_height` wird die Höhe und Breite des Buttons beschrieben. Die Option `fill_parent` lässt den Button über die ganze Breite des Bildschirmes anzeigen, abhängig von der Auflösung des Endgerätes. Die Option `wrap_content` lässt den Button nur so groß werden, dass alle Inhalte gut zu erkennen sind.
Parallel zu der grafischen Gestaltung der Activity `Register.xml` wird die eigentliche Logik in Java-Klassen ausgelagert, um eine strikte Trennung von GUI und Fachlogik zu erreichen. Die Klasse `SignUp.java` ist diesem Fall die Klasse die sich auf das XML-File `Register.xml` referenziert. Um die einzelnen Objekte des Layouts ansprechen zu können, werden im ersten Schritt alle einzelnen Komponenten erzeugt und im Nachhinein mit der Funktion `Objekt.findViewByID.ObjektID` auf das jeweilige gerade erzeugte Objekt in der Java-Klasse referenziert. Um überhaupt eine funktionierende Activity zu programmieren brauch man die Funktion `onCreat()`. In dieser Funktion wird all das ausgeführt was beim starten der Activity passieren soll. Zum einen das Referenzieren

auf die erzeugten Objekte und weitere Funktionsaufrufe. Diese Daten werden alle in ein JSON-Objekt geschrieben und per POST-Methode an den Server gesendet. Auf dem Server wird in der Index.php erkannt, dass sich ein neuer Nutzer anmelden möchte dem entsprechend wird in einer Mehrfachauswahl (Switch-Case) ausgewertet und in einem weiteren PHP-Skript weiter bearbeitet. Schlussendlich werden die Daten in eine MySQL-Datenbank in die jeweiligen Spalten geschrieben. Das eingegebene Passwort beim Eintragen in die Datenbank verschlüsselt, dass mögliche Hacker die das Password nicht auslesen können.

Der eigentliche Login

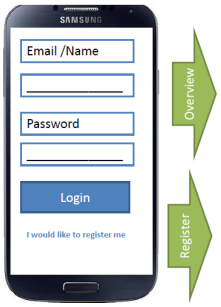

Abbildung 3.3: Login

Beim Login verläuft der Vorgang wie bei der eben beschriebenen Registrierung, bloß in die andere Richtung. Dabei wird ein JSON-Objekt vom Server an das mobile Endgerät geschickt und in der App aufgeschlüsselt und interpretiert. Danach wird verglichen ob sich der Nutzer der sich gerade einloggen möchte schon eingetragen ist, wenn ja wird auf die nächste Activity weitergeleitet, ansonsten wird er zur Registrierung geführt und gebeten sich anzumelden.

3.1.3 Control View

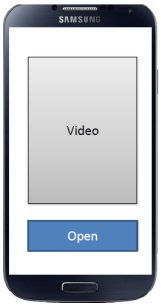

Abbildung 3.4: Control View

3.1.4 Datenbank

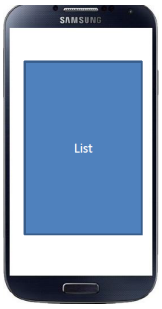

Abbildung 3.5: Datenbank

Kapitel 4

Desktop App

Zu der Android App gibt es parallel eine App für den Desktop. Diese wurde mit JavaFX programmiert. In den folgenden Kapiteln wird die ungefähre Programmierung der einzelnen Stages beschrieben.

4.1 Was ist JavaFX ?

JavaFX ist eine Java Spezifikation die als Hauptkonkurrenten Adobe Flash und Microsoft Silverlight hat. Ein positiver Punkt ist der Lauffähigkeit auf diversen Geräten wie z.B. Mobilfunk, Desktop-Computern, Embedded Geräten und Blu-ray Geräten. Die Programmierung findet ganz normal wie in Java statt. Die dazu gehörigen Bibliotheken werden seit der Java SE Runtime 7 Update 6 automatisch mit installiert. Es ist unter anderem auf die Grafikprogrammierung ausgelegt. Dadurch lassen sich Grafische Elemente schnell programmieren und mit CSS gestalten.(Quelle: [**?**]) Ein sehr bekanntes Embedded Gerät wofür es auch JavaFX gibt ist das Raspberry Pi.

4.2 Struktur und Aufbau der App

Es gibt insgesamt 5 verschiedene Stages in der App.

- Login (siehe **??**)
- Registrierung (siehe **??**)
- Control (siehe **??**)
- Datenbank (siehe **??**)
- Foto (siehe **??**)

In JavaFx ist ein solches Fenster ein Stage Objekt. Diesem Stage Objekt können mehrere anderer Objekte hinzugefügt werden. Bei diesen anderen Objekten kann es sich um Buttons, eine Tabelle, ein Textfeld usw handeln.

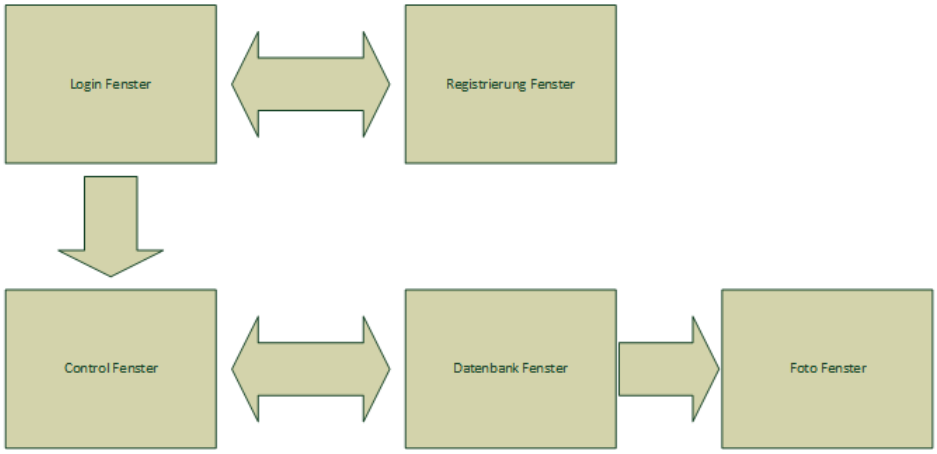

Abbildung 4.1: Aufbau der App

4.2.1 Login

Bei dem Login Fenster muss sich der Nutzer mit seinem Usernamen und Passwort was in der Datenbank hinterlegt ist anmelden. Ist einer der Felder nicht ausgefüllt oder das Passwort bzw der User nicht korrekt wird das mit einem `Label` als Message dargestellt. Über eine `CheckBox` kann der Benutzer sich sein Passwort in Klartext anzeigen oder verbergen lassen. Dies wurde so realisiert das ein `TextField` Objekt und ein `PasswordField` Objekt direkt übereinander gelegt wurden. Der Initialisierungszustand ist der das das `PasswordField` sichtbar und das `TextField` unsichtbar ist. Mit der `CheckBox` wird das ganze dann getoggelt.

```
    pwTextField.managedProperty().bind(checkBox.selectedProperty());
    pwTextField.visibleProperty().bind(checkBox.selectedProperty());

    passwordField.managedProperty().bind(checkBox.selectedProperty().not());
    passwordField.visibleProperty().bind(checkBox.selectedProperty().not());
```

Listing 4.1: Java Passwort-, Textfeld Un-, Sichtbar

Damit das eingegeben mit in dem anderen Feld erscheint werden beide Felder bidirektional miteinander verbunden.

```
    pwTextField.textProperty().bindBidirectional(
            passwordField.textProperty());
```

Listing 4.2: Java Passwort-, Textfeld bidirektional

Abbildung 4.2: Login Fenster

4.2.2 Registrierung

Falls der Benutzer noch kein Konto in der Datenbanktabelle `tb_Users` hat, hat er die Möglichkeit sich über das Registrierung Fenster anzumelden. Software seitig wurde eine Überprüfung eingebaut das jedes Textfeld etwas beinhaltet. Bei den Passwortfeldern wird mit überprüft ob die beiden Passwörter identisch sind. Wenn alle Überprüfungen korrekt sind wird aus den Eingaben und einem SQL Befehl `NOW()` ein SQL-String gebaut und an die Datenbank geschickt. Falls die Überprüfung fehlgeschlagen ist wird, wie im Login Fenster (s. **??**) eine `Label` Message ausgegeben.

```
String SQL = "INSERT␣INTO␣tb_user␣VALUES␣(null,'"
                + txtVorname.getText() + "',␣'"
                + txtNachname.getText() + "',␣'"
                + txtEmail.getText() + "',␣'"
                + txtUserName.getText() + "',␣'"
                + txtPw.getText() + "',␣NOW())";
```

Listing 4.3: Java-SQL neuer Benutzer

Der folgende String zeigt die Darstellung wie der obige String mit Nutzerdaten aussieht und an die Datenbank gesendet wird.

```
INSERT INTO tb_user VALUES (null,'Gernot', 'Hassknecht', 'heutShow@zdf.de', 'Hassi',
```

Listing 4.4: SQL Beispiel String

Der Befehle `NOW()` wird in der Datenbank mit dem aktuellen Datum und Uhrzeit ersetzt. Die Sichtbarkeit der Anmeldedaten ist nur direkt innerhalb der Applikation möglich. In diesem Fall mit einem System.out.println() in Eclipse. Was nicht mehr weiter implementiert wurde ist eine extra freischaltung des neuen Users von einem Admin. Nach direkter Registrierung kann sich der User sofort anmelden.

Abbildung 4.3: Registrierung Fenster

4.2.3 Control

4.2.4 Datenbank

Diese Stage hat als Hauptobjekt eine Tabelle. Der Tabelleninhalt wird dynamisch erstellt. In einer extra Klasse wird geschaut wie viele Spalten die Datenbank hat und fügt diese dann dem Tabellen

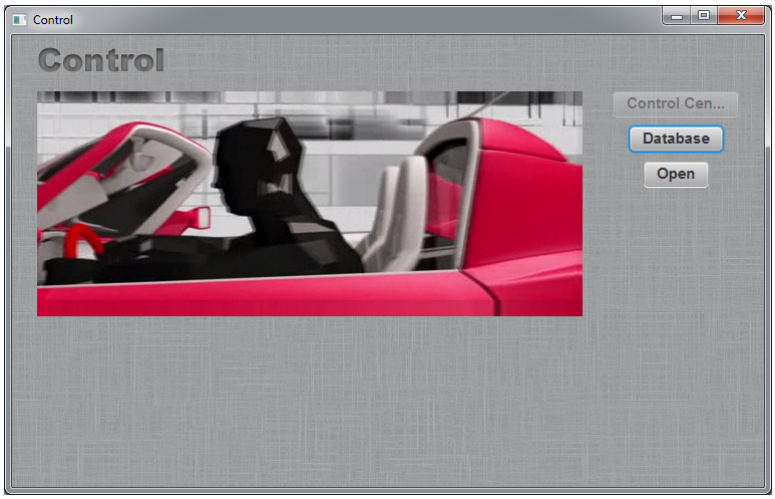

Abbildung 4.4: Control Fenster

Objekt hinzu. Nach dem hinzufügen der Spalten wird Zeile für Zeile aus der Datenbank geholt und in die Tabelle geladen. Zu jedem Eintrag in die Datenbanktabelle *tb_doorlogger* gehört ein Bild. Um sich zu einen entsprechenden Tabelleneintrag das Bild anzusehen muss man über eine Combobox die ID der Zeile auswählen und auf den Button *Open Picture* klicken. Mehr dazu im Kapitel **??**. Die Combobox zeigt nur so viele Zahlen wie es Zeilen in der Tabelle gibt. Wenn nichts in der Datenbank steht und dadurch auch kein Eintrag in die Tabelle gemacht wird, werden die Combobox und der *Open Picture* Button deaktiviert.

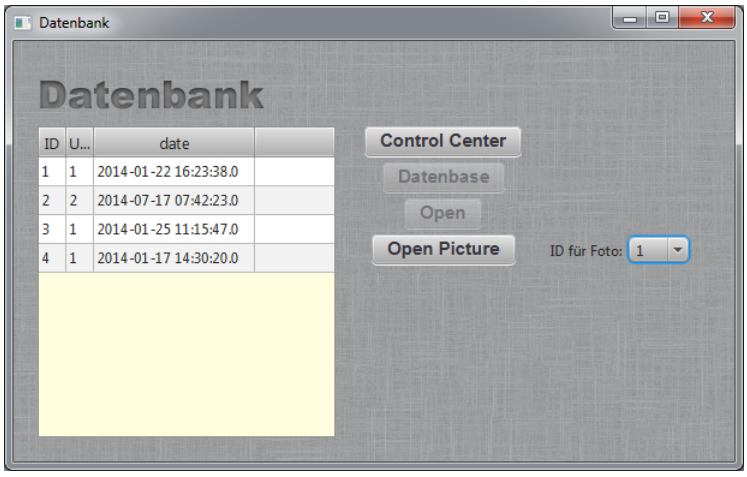

Abbildung 4.5: Datenbank Fenster

4.2.5 Foto

Nachdem der `Open Picture` Button in der Datenbank Ansicht gedruckt wurde wird mit Hilfe der angegebenen ID aus der Combobox der SQl String gebaut.

```
String SQL = "SELECT * FROM tb_images WHERE ID = " + userID;
```

Listing 4.5: Java-SQL String Foto öffnen

Aus dem String ist relativ leicht zuerkennen das das Bild aus der Datenbanktabelle `tb_images` kommt. Das Bild ist aber in der Datenbank nur Binär abgelegt. Als BLOB Typ. Dieser Type existiert auch in Java. Nach dem auslesen des Binärstreams wandeln wir das gelesene in ein Byte Array. Aus dem Byte Array erzeugen wir dann ein Buffered Image. In Swing könnten wir jetzt schon ein Bild uns anzeigen lassen. Aber das Programm wurde ja nicht mit Swing sondern mit JavaFX geschrieben. Dank eines `.toFXImage(BufferedImage arg0, WritableImage arg1)` Befehles lässt sich unser Swing Objekt einfach in ein JavaFX Objekt umwandeln. Dieses zeigen wir dann an. Ein klarer Vorteil bei dieser Methode ist das es nicht wichtig ist was für ein Typ dem Bild mal angehörte.

```
//Foto aus DB holen
byte[] imgData = imgShow.getImageDB(userID);
...
//Foto nach JavaFX Objekt wandeln
SwingFXUtils.toFXImage(bufImg, img2);

//Foto imageView hinzufügen
imageView.setImage(img2);
```

Listing 4.6: JavaFX Foto öffnen

Kapitel 5

Zusammenführung und Ausblick

Abbildungsverzeichnis

Listings

Anhang A

Anhang

Literaturverzeichnis