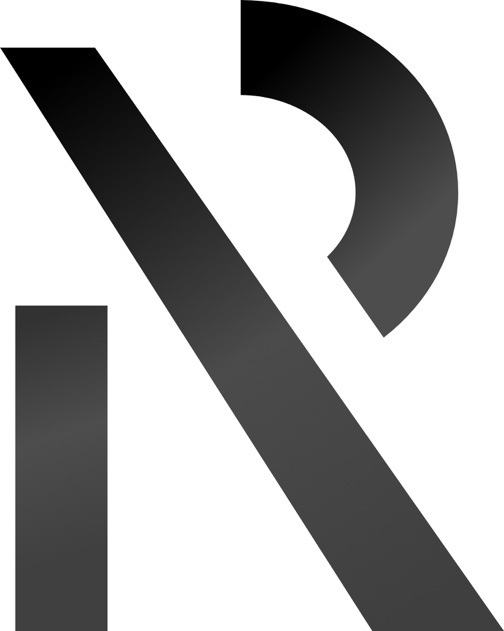


HTL_Logo**HTBLuVA Salzburg**

**Höhere Lehranstalt für**

**Elektronik und Technische Informatik**

**DIPLOMARBEIT**



Gesamtprojekt

**Reflecton**

**Entwicklung eines intelligenten Spiegels**

Marco Hennermann 5CHEL

Alexander Jeitler-Stehr 5CHEL

Lukas Korntner 5CHEL

Betreuer: AV Ing. Dipl.-Ing. Karl Heinz Steiner

Kooperationspartner: Glas Schnabl

ausgeführt im Schuljahr 2017/18

Abgabevermerk:

Datum: 23.03.2018 übernommen von:

**Eidesstattliche Erklärung**

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 23.03.2018 Verfasserinnen / Verfasser:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vorname Nachname

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vorname Nachname

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vorname Nachname

**DIPLOMARBEIT**

**DOKUMENTATION**

|  |  |
| --- | --- |
| Namen der Verfasser | Marco Hennermann, Alexander Jeitler-Stehr, Lukas Korntner |
| Jahrgang / Schuljahr | 5CHEL |
| Thema der Diplomarbeit | Reflecton – Entwicklung eines intelligenten Spiegels |
| Kooperationspartner | Glas Schnabl |

|  |  |
| --- | --- |
| Aufgabenstellung | Es ist ein Spiegel zu entwickeln, welcher über einen Raspberry Pi alltägliche Informationen wie Wetter, E-Mails oder Kalendereinträge auf einem LCD-Display anzeigt. Der Spiegel soll mittels Gestensteuerung bedient und per Smartphone-App sowohl auf iPhones als auch auf Android-Geräten eingerichtet und individualisiert werden können. |
| Realisierung | Der Grundgedanke unseres Projekts war die beste Lösung für die Bedienung zu finden. Da ein Touchscreen auf einem Spiegel nur zum Verschmieren führt, haben wir uns für eine Gestensteuerung entschieden. Mit jener Steuerung können Benutzer ganz einfach in die gewünschte App springen. Ein weiterer wichtiger Punkt sind die Vernetzungsmöglichkeiten von Reflecton. Er kann vollautonom über eine App am Smartphone eingerichtet werden. Mit dieser können die Benutzer nahtlos ihren Spiegel personalisieren, ihre Apps anordnen und ihre Email-/Kalender-Accounts hinzufügen. Die Einrichtung des Internets am Spiegel ist ebenso einfach wie die intelligente Bedienung. Das Smartphone verbindet sich automatisch über einen Hotspot mit dem Reflecton-Spiegel, wodurch ganz einfach das WLAN und das zugehörige Passwort, welches Reflecton verwenden soll, übertragen werden. Besonders hervorzuheben ist, dass unser Team einen großen Wert auf Design gelegt hat. Einen entscheidenden Teil trägt der aus edlem Eichen-Vollholz gefertigte Rahmen dazu bei, in welchem das Display hinter einem Spionspiegel steckt. |
| Ergebnisse | Zusammenfassend können wir mit Stolz behaupten, alle Musskriterien erfüllt zu haben. Außerdem wurde jegliche Art von Problemen rasch behoben. Hinzu kamen einige Erweiterungen, welche wir aus zeitlichen Gründen beginnen und größtenteils sogar abschließen konnten. Des Weiteren eignet sich unser Projekt hervorragend zur Weiterführung. Es könnten noch viele weitere Widgets wie beispielsweise Facebook, WhatsApp, oder YouTube hinzugefügt werden. |

|  |  |
| --- | --- |
| Projektspezifische Grafik |  |

|  |  |
| --- | --- |
| Teilnahme an Wettbewerben und Auszeichnungen | * Jugend Innovativ |

|  |  |
| --- | --- |
| Möglichkeiten der Einsichtnahme in die Arbeit | * Bibliothek HTL-Salzburg * Infoserver HTL-Salzburg |

|  |  |  |
| --- | --- | --- |
| Approbation (Datum / Unterschrift) | Prüfer  AV Ing. Dipl.-Ing Karl Heinz Steiner | Direktor  Dir. Dipl.-Ing Dr. Andreas Magauer |

**DIPLOMA-THESIS**

**DOCUMENTATION**

|  |  |
| --- | --- |
| Authors | Marco Hennermann, Alexander Jeitler-Stehr, Lukas Korntner |
| Academic year | 5CHEL |
| Topic | Reflecton – Development of an intelligent mirror |
| Co-operation Partners | Glas Schnabl |

|  |  |
| --- | --- |
| Assignment of Tasks | The target was to develop a smart mirror, which sends personal informations like the local weather report or recently received e-mails to a mini-computer. Therefor a connection to a database needs to be established. The table, located in the database, contains the data of all accounts, which got registered through our smart phone app. This application is available on iOS as well as on Android. It enables the user to configure the mirror and individualize the apps on the mirror. |
| Realization | The fundamental thought of our project was to get the best solution for controlling the mirror. Because a touchscreen would cause a smeary film on the mirror, we decided to tackle this via gesture control. With this control system the costumer can easily jump into the targeted app. Another crucial point is the connectivity of Reflecton. It is fully able to be controlled automatically by the smart phone-app. With this app, the user can personalize the mirror, sort the widgets and add email- and calendar-accounts. The configuration of an internet connection is as easy as the intelligent controlling. The cell phone connects automatically over a hotspot with the Reflecton-mirror. Because of that, the WiFi and its password get transmitted. Especially worth to mention is the fact that our team sets a high value on design. One decisive part of this is the frame which is made of oak solid wood. It sheathes the display which hides behind a spy mirror. |
| Results | Despite all obstacles and challenges, we have managed to implement all must-haves in the way we targeted. The concept has even more extension possibilities. One of them are additional apps like Facebook, WhatsApp, YouTube or the news. |

|  |  |
| --- | --- |
| Project-specific Illustration |  |

|  |  |
| --- | --- |
| Participation in Competitions and Awards | * Jugend Innovativ |

|  |  |
| --- | --- |
| Accessibility of Diploma Thesis | * Library HTL-Salzburg * Information-Server HTL-Salzburg |

|  |  |  |
| --- | --- | --- |
| Approval (Date / Sign) | Examiner  AV Ing. Dipl.-Ing Karl Heinz Steiner | Head of College  Dir. Dipl.-Ing Dr. Andreas Magauer |

**Vorwort**

Bei Gesprächen innerhalb der Klasse war zu erkennen, dass diejenigen, welche mit dem Auto zur Schule fahren, über personenspezifische Informationen in geringerer Genauigkeit verfügten als diejenigen, welche mit öffentlichen Verkehrsmitteln unterwegs sind. Dabei kam uns der Gedanke, dass Autofahrer allgemein nur über zusätzlichen Zeitaufwand schon vor dem in die Schule bzw. zur Arbeitsstelle kommen am Handy oder in der Tageszeitung nach für die Person interessanten Dingen forschen können, während ein Bus- oder Zugfahrer diesen Nachteil nicht hat.

Da auch heutzutage noch ein großer Teil mit dem Auto zum Arbeitsplatz bzw. zur Ausbildungsstelle fährt, war uns damit klar, dass wir etwas für diese Bevölkerungsgruppe entwickeln wollen, um diese Benachteiligung aufzuheben. Es gibt zwar schon potenzielle Lösungen für dieses Problem, jedoch sind die Kosten für derartige Produkte fast ausschließlich im vierstelligen Euro-Bereich und damit für die meisten uninteressant. Unsere Idee für die Füllung dieser Marktlücke war ein halbdurchlässiger Spiegel, hinter welchem sich ein Display versteckt. Auf diesem Display werden Daten wie der aktuelle Wetterbericht oder Kalendereinträge angezeigt. Diese werden per Smartphone-App individualisierbar und per Wischgesten anwählbar.

In diesem Zusammenhang möchten wir uns bei Herrn AV Ing. Dipl.-Ing. Karl Heinz Steiner und dem Unternehmen Glas Schnabl besonders für ihre Unterstützung bedanken.

**Abstract**

While talking in our school we recognized that the students who usually get to school by car have a clear disadvantage against the ones who travel by bus or train when it comes to knowledge about daily relevant information. The main reason for this is that one who drives a car would only be able to get useful information in front of a working day in connection with some amount of time investment. The most common examples are checking the smart phone or reading newspapers or magazines. It was this moment we came up with the idea that a serious solution to this problem would probably be very useful for the world, because even nowadays the main part of humanity drives to work by car. Although we found some products, which potentially solve this issue, we still recognized why only a little part of society was interested in these products. Every solution we found was too expensive to hit the consumer market in a positive way. From this moment on the way was clear for starting to make our own concept to solve this problem with pushing the price down and though keeping the quality on an elevated level. First, this concept contents a semi-transparent mirror. A lcd-display hides behind that. On this display, personal information like the local weather report or received e-mails are getting visualized. A clever gesture control system allows to get more detailed data with one little move. A smart phone for the main operating systems iOS and Android enables remote configuration and individualization.

In this context we would like to thank the Head of Department, Ing. Dipl.-Ing. Karl Heinz Steiner and the company of Glas Schnabl for their support.

**Inhaltsverzeichnis**

[**1.** **Einleitung** 13](#_Toc509558413)

[**2.** **Aufgabenstellung** 14](#_Toc509558414)

[**2.1** **Projektorganisation** 14](#_Toc509558415)

[**2.2** **Aufgabenbeschreibung** 14](#_Toc509558416)

[**2.3** **Zielsetzung** 14](#_Toc509558417)

[**2.4** **Konzept** 15](#_Toc509558418)

[**3.** **Komplettsystem** 16](#_Toc509558419)

[**4.** **Konstruktion** 17](#_Toc509558420)

[**4.1** **Konstruktion des Rahmens** 17](#_Toc509558421)

[**4.2** **Rendering des Rahmens** 19](#_Toc509558422)

[**4.3** **Displayschablone** 21](#_Toc509558423)

[**4.4** **Spiegelglas** 21](#_Toc509558424)

[**4.5** **Deckel** 22](#_Toc509558425)

[**4.6** **Wandhalterung** 22](#_Toc509558426)

[**5.** **Ubuntu-Server Setup** 23](#_Toc509558427)

[**5.1** **Installation der benötigten Software** 23](#_Toc509558428)

[**5.2** **Server Login** 23](#_Toc509558429)

[**5.3** **Aktualisierung des Servers** 24](#_Toc509558430)

[**5.4** **Installation des Apache-Servers** 25](#_Toc509558431)

[**5.5** **Installation von MariaDB** 26](#_Toc509558432)

[**5.6** **Installation von PHP-7** 28](#_Toc509558433)

[**6.** **MySQL-Datenbank** 30](#_Toc509558434)

[**6.1** **Verwaltung der MySQL Datenbank** 30](#_Toc509558435)

[6.1.1 Neuen Benutzer in der Datenbank erstellen 30](#_Toc509558436)

[6.1.2 Erstellung der Datenbank in PHPMyAdmin 31](#_Toc509558437)

[**7.** **PHP-Interface zur Datenbank** 34](#_Toc509558438)

[**7.1** **Warum PHP?** 34](#_Toc509558439)

[**7.2** **Logik** 34](#_Toc509558440)

[**7.3** **Struktur** 35](#_Toc509558441)

[**7.4** **Konfigurationsdateien** 35](#_Toc509558442)

[7.4.1 Reflecton/includes/config.php 36](#_Toc509558443)

[7.4.2 Reflecton/includes/signup\_config.php 36](#_Toc509558444)

[7.4.3 Reflecton/includes/signin\_config.php 36](#_Toc509558445)

[7.4.4 Reflecton/includes/setdata\_config.php 37](#_Toc509558446)

[7.4.5 Reflecton/includes/getdata\_config.php 37](#_Toc509558447)

[**7.5** **Klassen** 37](#_Toc509558448)

[7.5.1 Reflecton/includes/Column.php 38](#_Toc509558449)

[7.5.2 Reflecton/includes/Database.php 39](#_Toc509558450)

[**7.6** **Schnittstellendateien** 45](#_Toc509558451)

[7.6.1 Reflecton/api/signup.php 46](#_Toc509558452)

[7.6.2 Reflecton/api/signin.php 47](#_Toc509558453)

[7.6.3 Reflecton/api/setdata.php 47](#_Toc509558454)

[7.6.4 Reflecton/api/getdata.php 48](#_Toc509558455)

[**7.7** **HTML-Interface** 48](#_Toc509558456)

[7.7.1 Reflecton/includes/interface\_signin.php 48](#_Toc509558457)

[**8.** **PHP-Interface zum Pi** 50](#_Toc509558458)

[**8.1** **Vorbereitungen** 50](#_Toc509558459)

[8.1.1 Vorbereitungen am Raspberry Pi 50](#_Toc509558460)

[8.1.2 Vorbereitungen für Hotspot 51](#_Toc509558461)

[8.1.3 Auto-Hotspot Skript 55](#_Toc509558462)

[8.1.4 Vorbereitungen für PHP-Interface 58](#_Toc509558463)

[**8.2** **WLAN-Konfiguration mithilfe von PHP** 58](#_Toc509558464)

[**9.** **User Interface** 61](#_Toc509558465)

[**9.1** **Struktur** 61](#_Toc509558466)

[**9.2** **Vorbereitungen** 62](#_Toc509558467)

[9.2.1 Raspberry Pi Konfiguration 62](#_Toc509558468)

[9.2.2 Java Update 63](#_Toc509558469)

[9.2.3 Implementierung von Pi4J 64](#_Toc509558470)

[9.2.4 Remote Platform 65](#_Toc509558471)

[9.2.5 Modifizierung der build.xml 67](#_Toc509558472)

[9.2.6 Bildschirm rotieren 68](#_Toc509558473)

[**9.3** **Hinzufügen der Bibliotheken** 69](#_Toc509558474)

[**9.4** **Konfigurationsdateien** 70](#_Toc509558475)

[9.4.1 ReflectonUI/com/pinnovations/config/DatabaseConfig.java 70](#_Toc509558476)

[9.4.2 ReflectonUI/com/pinnovations/config/FileConfig.java 71](#_Toc509558477)

[9.4.3 ReflectonUI/com/pinnovations/config/FontConfig.java 71](#_Toc509558478)

[9.4.4 ReflectonUI/com/pinnovations/config/ImageConfig.java 71](#_Toc509558479)

[**9.5** **Datenbankspezifische Dateien** 72](#_Toc509558480)

[9.5.1 ReflectonUI/com/pinnovations/database/Column.java 72](#_Toc509558481)

[9.5.2 ReflectonUI/com/pinnovations/database/Database.java 72](#_Toc509558482)

[9.5.3 ReflectonUI/com/pinnovations/database/InterfaceDatabase.java 76](#_Toc509558483)

[**9.6** **Downloadspezifische Dateien** 76](#_Toc509558484)

[9.6.1 ReflectonUI/com/pinnovations/download/Appointment.java 77](#_Toc509558485)

[9.6.2 ReflectonUI/com/pinnovations/download/Clock.java 77](#_Toc509558486)

[9.6.3 ReflectonUI/com/pinnovations/download/Day.java 78](#_Toc509558487)

[9.6.4 ReflectonUI/com/pinnovations/download/Email.java 79](#_Toc509558488)

[9.6.5 ReflectonUI/com/pinnovations/download/GeoLocation.java 80](#_Toc509558489)

[9.6.6 ReflectonUI/com/pinnovations/download/InterfaceDownload.java 81](#_Toc509558490)

[**9.7** **Hilfsdateien** 88](#_Toc509558491)

[9.7.1 ReflectonUI/com/pinnovations/filereader/Filereader.java 88](#_Toc509558492)

[9.7.2 ReflectonUI/com/pinnovations/support/FontSupport.java 89](#_Toc509558493)

[9.7.3 ReflectonUI/com/pinnovations/support/TableSupport.java 90](#_Toc509558494)

[9.7.4 ReflectonUI/com/pinnovations/exceptions/FieldNotFoundException.java 91](#_Toc509558495)

[**9.8** **Widgets** 92](#_Toc509558496)

[9.8.1 ReflectonUI/com/pinnovations/widgets/Widget.java 92](#_Toc509558497)

[9.8.2 ReflectonUI/com/pinnovations/widgets/EmailWidget.java 93](#_Toc509558498)

[9.8.3 ReflectonUI/com/pinnovations/widgets/CalendarWidget.java 95](#_Toc509558499)

[9.8.4 ReflectonUI/com/pinnovations/widgets/WeatherWidget.java 97](#_Toc509558500)

[9.8.5 ReflectonUI/com/pinnovations/widgets/ClockWidget.java 99](#_Toc509558501)

[**9.9** **Ansichten** 101](#_Toc509558502)

[9.9.1 ReflectonUI/com/pinnovations/views/MainView.java 101](#_Toc509558503)

[9.9.2 ReflectonUI/com/pinnovations/views/EmailView.java 106](#_Toc509558504)

[9.9.3 ReflectonUI/com/pinnovations/views/CalendarView.java 108](#_Toc509558505)

[9.9.4 ReflectonUI/com/pinnovations/views/WeatherView.java 110](#_Toc509558506)

[9.9.5 ReflectonUI/com/pinnovations/views/ClockView.java 111](#_Toc509558507)

[**9.10** **Hauptframe und Hauptklasse** 112](#_Toc509558508)

[9.10.1 ReflectonUI/com/pinnovations/frames/MainFrame.java 112](#_Toc509558509)

[9.10.2 ReflectonUI/com/pinnovations/main/ReflectonUI.java 113](#_Toc509558510)

[**10.** **Gestensteuerung** 116](#_Toc509558511)

[**10.1** **Vor- und Nachteile** 116](#_Toc509558512)

[**10.2** **Potenzielle Varianten und deren Nachteile** 117](#_Toc509558513)

[10.2.1 MGC3x30 GestIC 117](#_Toc509558514)

[10.2.2 Leap Motion 118](#_Toc509558515)

[10.2.3 Skywriter HAT 118](#_Toc509558516)

[10.2.4 ZX Gesture Detection 118](#_Toc509558517)

[10.2.5 Gestenbibliotheken 119](#_Toc509558518)

[**10.3** **OpenCV** 120](#_Toc509558519)

[10.3.1 Beschreibung der Gestenbibliothek 120](#_Toc509558520)

[10.3.2 NoIR-Pi-Kamera mit Gesten-Bibliothek 120](#_Toc509558521)

[10.3.3 Gesteninterpretationsprogramm 121](#_Toc509558522)

[**10.4** **APDS-9960** 126](#_Toc509558523)

[10.4.1 Allgemeine Beschreibung 126](#_Toc509558524)

[10.4.2 Verdrahtung am Arduino 128](#_Toc509558525)

[10.4.3 Gesteninterpretation 129](#_Toc509558526)

[10.4.4 GP2Y0A02YK0F-Infrarotsensor als Erweiterung 133](#_Toc509558527)

[**11.** **iOS-App** 134](#_Toc509558528)

[**11.1** **Storyboards** 134](#_Toc509558529)

[**11.2** **Workflow** 135](#_Toc509558530)

[**11.3** **Struktur** 136](#_Toc509558531)

[**11.4** **Konfigurationsdatei** 136](#_Toc509558532)

[**11.5** **Designspezifische Dateien** 137](#_Toc509558533)

[11.5.1 Reflecton/Layouts/WaterfallCollectionViewLayout.swift 137](#_Toc509558534)

[11.5.2 Reflecton/Views/TextCollectionViewCell.swift 137](#_Toc509558535)

[11.5.3 Reflecton/CustomSegues/SegueToLeft.swift 138](#_Toc509558536)

[11.5.4 Reflecton/CustomElements/DesignableTextField.swift 139](#_Toc509558537)

[**11.6** **Ansichten** 140](#_Toc509558538)

[11.6.1 Reflecton/ViewControllers/SignUpViewController.swift 140](#_Toc509558539)

[11.6.2 Reflecton/ViewControllers/SignInViewController.swift 145](#_Toc509558540)

[11.6.3 Reflecton/ViewControllers/HomeViewController.swift 149](#_Toc509558541)

[11.6.4 Reflecton/ViewControllers/WiFiViewController.swift 158](#_Toc509558542)

[**12.** **Android-App** 163](#_Toc509558543)

[**12.1** **Vorbereitung** 163](#_Toc509558544)

[12.1.1 Erstellungsdetails 163](#_Toc509558545)

[12.1.1 Design 164](#_Toc509558546)

[**12.2** **Programmierung** 164](#_Toc509558547)

[12.2.1 Start-Anwendung 164](#_Toc509558548)

[12.2.2 Datenbankverbindung 166](#_Toc509558549)

[12.2.3 Registrierungsanwendung 167](#_Toc509558550)

[12.2.4 Widgets-Anwendung 169](#_Toc509558551)

[12.2.5 Einstellungen-Anwendung 170](#_Toc509558552)

[**12.3** **Inbetriebnahme** 171](#_Toc509558553)

[**13.** **Webseite** 172](#_Toc509558554)

[**13.1** **Startseite** 172](#_Toc509558555)

[13.1.1 Startseite in HTML 172](#_Toc509558556)

[13.1.2 Startseite in CSS 173](#_Toc509558557)

[**13.2** **Log In / Sign Up** 176](#_Toc509558558)

[13.2.1 Log In / Sign Up in HTML 176](#_Toc509558559)

[13.2.2 Log In / Sign Up in CSS 178](#_Toc509558560)

[13.2.3 Log In / Sign Up in Javascript 178](#_Toc509558561)

[13.2.4 Log In / Sign Up in PHP 179](#_Toc509558562)

[**13.3** **Interner Bereich** 182](#_Toc509558563)

[13.3.1 Interner Bereich in HTML 182](#_Toc509558564)

[13.3.2 Interner Bereich in CSS 182](#_Toc509558565)

[13.3.3 Interner Bereich in Javascript 183](#_Toc509558566)

[13.3.4 Interner Bereich PHP 184](#_Toc509558567)

[**14.** **Ergebnis** 185](#_Toc509558568)

[**15.** **Abbildungsverzeichnis** 186](#_Toc509558569)

[**16.** **Codeabschnitte** 188](#_Toc509558570)

[**17.** **Quellenverzeichnis** 193](#_Toc509558571)

[**18.** **Begleitprotokoll** 194](#_Toc509558572)

[**19.** **Anhang** 196](#_Toc509558573)

[**19.1** **Kostenschätzung** 196](#_Toc509558574)

[**19.2** **Gantt-Diagramm** 196](#_Toc509558575)

1. **Einleitung**

Das Konzept orientiert sich an dem Ziel, den Alltag des Menschen möglichst angenehm und unkompliziert zu gestalten. Unser intuitiver Spiegel kann maßgeblich dazu beitragen. Das Herzstück des Projekts stellt dabei ein LCD-Display, welches mit einer vorangestellten halbdurchlässigen Glaswand zum eleganten Spiegel wird. Dieses Display zeigt die Ausgabe eines Java-Programms, welches auf einem Raspberry Pi 3-B ausgeführt wird. Es werden sowohl allgemeine Informationen wie die Uhrzeit oder der lokale Wetterbericht als auch persönliche Informationen wie anstehende Termine oder empfangene E-Mails in den Ecken des Displays dargestellt. Mittels Gestensteuerung können die Detailansichten der einzelnen Anwendungen geöffnet werden. Dazu ist nur eine Wischgeste in die entsprechende Ecke notwendig. Des Weiteren wurde eine App zur Personalisierung der angezeigten Informationen sowohl für iOS- als auch für Android-betriebene Smartphones erstellt. In dieser muss einmalig ein Account erstellt werden, in welchem dann die einzelnen Anwendungen in ihrer Anordnung am Spiegel individuell verschoben werden können. All jene Informationen, die der Benutzer in der App eingibt, werden in einer Datenbank gespeichert und stehen in der Folge dem Java-Programm zur Verfügung.

1. **Aufgabenstellung**

Es ist ein Spiegel zu entwickeln, welcher über einen Raspberry Pi alltägliche Informationen wie Wetter, E-Mails oder Kalendereinträge auf einem LCD-Display anzeigt. Der Spiegel soll mittels Gestensteuerung bedient und per Smartphone-App sowohl auf iPhones als auch auf Android-Geräten eingerichtet und individualisiert werden können.

* 1. **Projektorganisation**

Das Reflecton-Projektteam:

* Projektleiter/-mitarbeiter: Alexander Jeitler-Stehr
* Projektmitarbeiter: Marco Hennermann
* Projektmitarbeiter: Lukas Korntner
* Projektbetreuer: AV Ing. Dipl.-Ing. Karl Heinz Steiner
  1. **Aufgabenbeschreibung**

Das Produkt soll dem Nutzer behilflich sein, die im Badezimmer verbrachte Zeit effizienter zu nutzen. Der Konsument soll sowohl persönliche als auch allgemeine Informationen am Spiegel dargestellt bekommen. Mit einfachsten Handbewegungen können Details zu gewissen Informationen aufgerufen werden. Am Handy soll eingestellt werden können, zu welcher E-Mail-Adresse die Informationen angezeigt werden sollen und welche Information in welchem Bereich dargestellt werden soll.

* 1. **Zielsetzung**
* Musskriterien
  + Datenbankverbindung
  + Programmierung der Benutzeroberfläche
  + iOS-App
  + Gestensteuerung
  + Aufbau des Prototyps
* Wunschkriterien
  + Android-App
  + Webseite
  1. **Konzept**

Von einer Smartphone-App aus kann sowohl per iPhone als auch per Android-Gerät der Spiegel konfiguriert werden. Dazu muss einmalig ein Konto inklusive E-Mail-Adresse angelegt werden. Die vom Smartphone aus eingegebenen Daten werden in einer Datenbank abgelegt. Am Spiegel ist ein Raspberry Pi 3B dafür zuständig, die dem Account entsprechenden Daten abzufragen. Um diesen Datenfluss zu bewerkstelligen, öffnet der Raspberry Pi einen Hotspot, mit welchem sich das Smartphone im Vorfeld verbinden muss. Erst dann bekommt der RPI Zugriff auf für ihn relevante Benutzerdaten wie die SSID oder die E-Mail-Adresse. Mit diesen Informationen können in der Folge E-Mail- und Kalender-Feed gestartet werden. Zusätzlich werden am Display Widgets mit der aktuellen Uhrzeit bzw. dem aktuellen lokalen Wetterbericht angezeigt. In der App kann außerdem die Anordnung dieser vier Widgets personalisiert werden. Dies wird auf dem mit dem Raspberry Pi verbundenen LCD-Display visualisiert. Dieses kommt zwar einem herkömmlichen Laptop-Display gleich, behält aber den Spiegeleffekt durch ein in Front angebrachtes halbdurchlässiges Spiegelglas bei, welches von einem Holzrahmen stilistisch ummantelt wird. In diesen Rahmen ist ein Infrarotsensor so implementiert, dass der Konsument ohne Mühe mit der Hand Wischbewegungen ausführen kann. Diese Wischbewegungen werden vom Sensor als Geste interpretiert, wodurch sich je nach dessen Richtung die Detailansicht des sich in der entsprechenden Ecke befindlichen Widgets öffnet. Die Interpretation findet am HTL UNO statt, welches den Befehl als serielle Ausgabe an das am Raspberry Pi laufende Java-Programm weitergibt.



Abbildung 1: Sketch des Komplettsystems

1. **Komplettsystem**

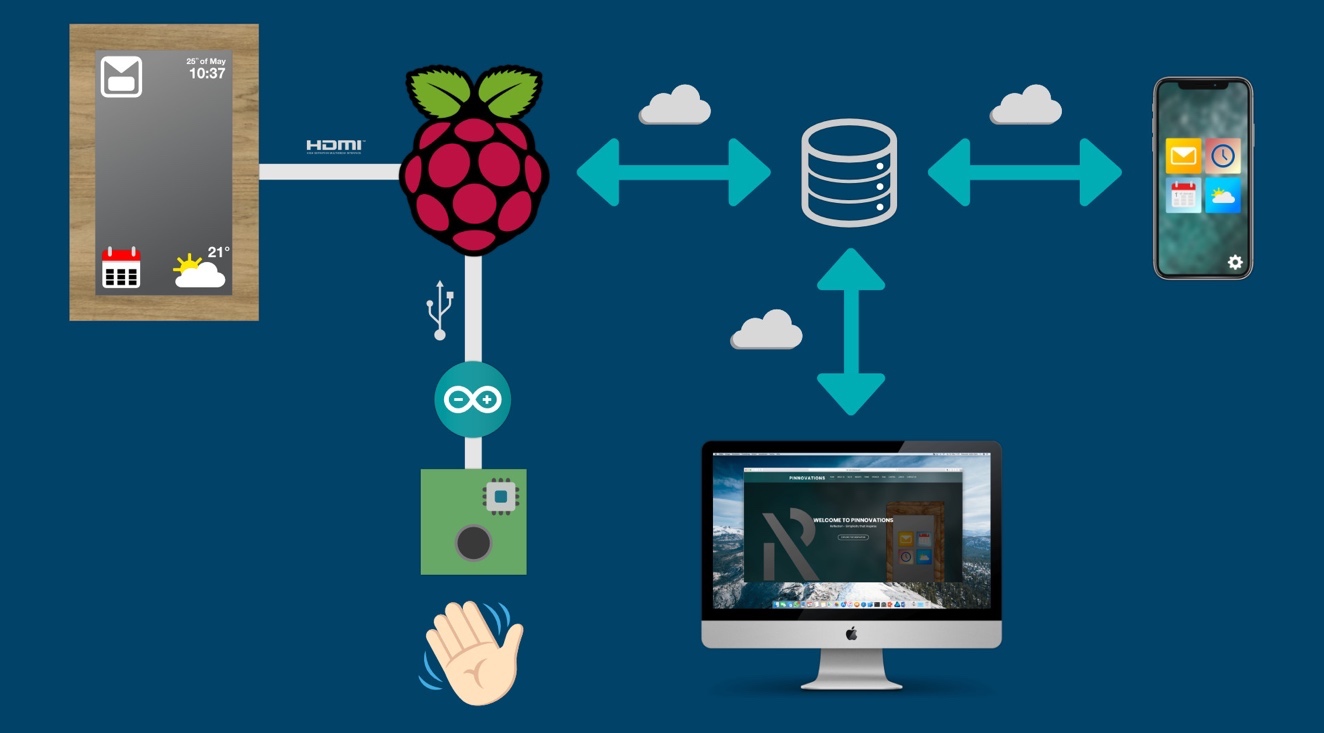


Abbildung 2: Blockschaltbild Komplettsystem

Zu Beginn ist der Raspberry PI mit keinem Netzwerk verbunden und strahlt einen eigenen Hotspot aus. Nun wird ein Smartphone mit dem WLAN-Netzwerk des Raspberry PI verbunden und es kann ein Benutzerkonto mithilfe der App auf dem Smartphone oder der Webanwendung erstellt werden. Nachdem der Benutzer eingeloggt ist, kann der Spiegel mit der App mit einem WLAN-Netzwerk verbunden werden. Hierbei werden die Zugangsdaten an den Raspberry PI übertragen und anschließend startet sich der Raspberry PI neu. Nun verbindet sich der Raspberry PI mit dem Netzwerk und die benötigten Daten für die Oberfläche werden heruntergeladen. Der Benutzer kann die Anwendungen mit seinen eigenen Benutzerkonten verbinden und die Anordnung der Anwendungen am Spiegel personalisieren. Wird eine Wischbewegungen in eine der vier Ecken vor dem Spiegel durchgeführt, so werden mithilfe des Arduino, welches den Gestensensor steuert, die erkannten Bewegungen an den Raspberry PI übertragen. Entsprechend diesen Bewegungen werden die Anwendung in den jeweiligen Ecken geöffnet. Mithilfe einer Bewegung nach links unten wird die aktuell geöffnete Anwendung geschlossen und in den Hauptbildschirm zurückgewechselt.

1. **Konstruktion**

Die verschiedenen Rahmen werden nun in einen Hauptrahmen aus Eichenholz eingebaut.

* 1. **Konstruktion des Rahmens**

Um zu sehen, wie der Rahmen in etwa aussehen sollte, wurde dieser zuerst mit Autodesk Fusion 360 entworfen. Dies ist eine kostenlose CAD Software von Autodesk.

Um eine neue Konstruktion starten zu können, wird ein neues Blatt in Autodesk Fusion 360 erstellt. Im Punkt *Dokumenteinstellungen*, sollte die Maßeinheit auf Millimeter gestellt werden.

|  |  |
| --- | --- |
| Abbildung 3: Grundrahmen | Im ersten Schritt wird in die Vorderansicht des Dokumentes gewechselt, um ein Rechteck mit den Abmessungen 350x550mm zu konstruieren (siehe Abbildung 3). Dies wird mit dem Linientool in der Kategorie der Skizzen realisiert. |

|  |  |
| --- | --- |
| Abbildung 4: Konstruktionsprozess des Rahmens | Um aus einem zweidimensionalen ein dreidimensionales Modell machen zu können, wird in die Hauptansicht zurückgewechselt. Mit dem Mauszeiger wird die umzuwandelnde Fläche angeklickt. Durch die Auswahl der *Extrusion* wird das Modell umgewandelt. Die Höhe des Rahmens beträgt 40mm, wie die Abbildung 4 deutlich zeigt. |

|  |  |
| --- | --- |
| Abbildung 5: Ausschnitt von der Rückseite des Rahmens | Nun wird dem Körper über das Kontextmenü über den Menüpunkt Darstellung eine Textur zugewiesen. In diesem Fall wird als Material Eiche ausgewählt, da der Rahmen später aus Eiche gefertigt wird. Zu finden ist diese Textur unter *Holz* 🡪 *unbehandelt*. Diese Textur sieht dem echten Holz sehr ähnlich. Auf der Rückseite des Körpers wird ein Rechteck in der Größe 300x500 mm gezeichnet. Dieses entspricht der Öffnung auf der Rückseite, in welchem die Elektronik und der Rahmen inklusive Display befestigt wird. Dieser Ausschnitt ist 35 mm (-35 mm da es in die entgegengesetzte Richtung der Achse verläuft) tief und über den Menüpunkt *Ändern* 🡪 *Drücken/Ziehen* angefertigt. Eine Wandstärke von 50mm ist somit gegeben, wie die nebenstehende Abbildung 4 zeigt. |

Zu guter Letzt wird das Sichtfenster für den Bildschirm eingezeichnet. Dieses Rechteck wird auf der Vorderseite des Rahmens mit einer Größe von 250x450 mm platziert. Das Sichtfenster wird gleich wie der Ausschnitt von der Rückseite angefertigt. Abbildung 6 zeigt die fertige Konstruktion.

|  |  |
| --- | --- |
| Abbildung 6: Konstruierter Rahmen | Um den erstellten Rahmen mit Archicad und Photoshop rendern zu können, muss diese Datei als *stl-*Datei abgespeichert werden. |

* 1. **Rendering des Rahmens**

|  |  |
| --- | --- |
| Abbildung 7: Rahmen in Archicad | In Archicad wird die nun eben erstellte Datei geöffnet. Wie die in Autodesk Fusion erstellte Datei mit Archicad geöffnet aussieht, zeigt Abbildung 7. |
| Abbildung 8: Einstellungen für Rendering | Es ist auch eine dreidimensionale Ansicht des Rahmens verfügbar. Diese kann über die Registerkarte erreicht werden.  Dieser Rahmen wird nun anschließend mithilfe eines sogenannten *Rendering*-Prozesses in eine fotorealistische Grafik umgewandelt. Von dem Rahmen wird in mehreren Ansichten ein Rendering erstellt, da somit ein Eindruck entsteht, wie der Rahmen nun von allen Ansichten aussieht. Die Einstellungen, welche in Abbildung 8 sichtbar sind, können unter dem Menüpunkt *Document* 🡪 *Creative Imaging* 🡪 *PhotoRendering Settings* gefunden werden.  Wird nun am unteren Rand des Fensters auf den Button geklickt, so startet der Rendering Prozess und der Rahmen Stück für Stück erstellt. Der Prozess des Rendering sieht, wie in Abbildung 9 sichtbar, aus. |

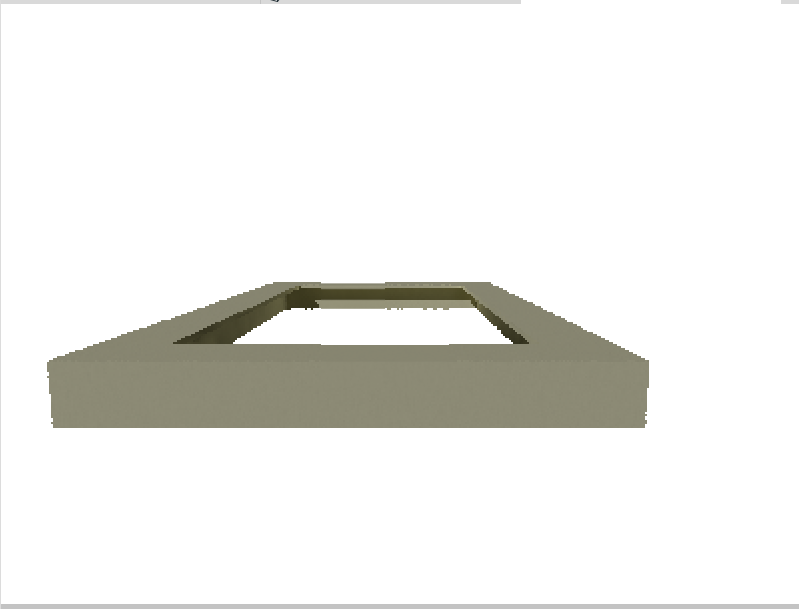


Abbildung 9: Rendering Prozess

Nun werden noch drei weitere *Renderings* mit denselben Einstellungen, allerdings anderer Ansichten durchgeführt. Wenn dieser Vorgang abgeschlossen ist, sollte das Ergebnis wie in Abbildung 10 und 11 aussehen. Als Textur wurde hierbei helles Fichtenholz ausgewählt.



Abbildung 10: Rahmen nach Rendering des Rahmens

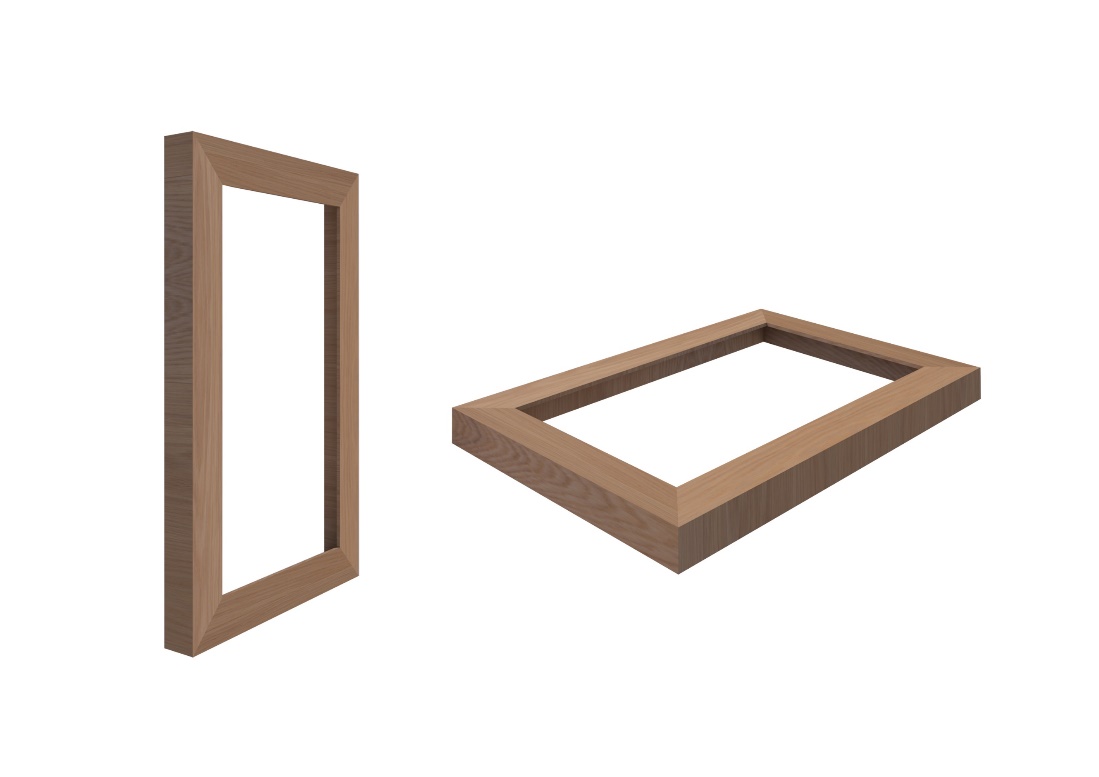


Abbildung 11: Rahmen nach dem durchgeführten Rendering

* 1. **Displayschablone**

Um die Größe des Displays an die des Rahmens anpassen zu können, wurde ein Rahmen aus PVC angefertigt. Dieser hat einen inneren Ausschnitt in den Abmessungen eines 17,3 Zoll großen Displays. An den Seiten wurde jeweils darauf geachtet, dass der PVC Rahmen nur minimal kleiner als der Holzrahmen ist, damit dieser auch ohne zusätzlicher Befestigung den Spiegel und das Display im Holzrahmen behält. Die Abbildung 12 zeigt, wie der PVC Rahmen aussieht.

.

Abbildung 12:Displayrahmen aus PVC

* 1. **Spiegelglas**

|  |  |
| --- | --- |
| Abbildung 13: Herstellerangaben Spiegel Glas (Quelle: Glas Schnabl) | Um die eigentliche Funktion eines Spiegels beizubehalten und trotzdem eine gewisse Lichtdurchlässigkeit zu haben, wurde in diesem Projekt ein Spiegel mit einem Lichttransmissionsgrad von 75% in den Abmessungen 300x500mm gewählt. Wenn das Display dahinter ausgeschaltet ist, reflektiert dieser wie ein handelsüblicher Spiegel. Die Abbildung 13 zeigt eine grafische Beschreibung des Spiegels von dem Hersteller. |

* 1. **Deckel**

|  |  |
| --- | --- |
| Abbildung 14: Hintere Abdeckung des Rahmens | Um zu verhindern, dass die Elektronik aus dem Rahmen herausfallen könnte, wurde der Luftbereich mit Styropor ausgefüllt und anschließend mit einer Kunststoffabdeckung abgeschlossen. Diese wurde etwas nach den Abmessungen kleiner dimensioniert, wie in der Abbildung 14 sichtbar ist. |

* 1. **Wandhalterung**

|  |  |
| --- | --- |
| Abbildung 15: Aufhängung des Spiegels | Damit der Spiegel später auch aufgehängt werden kann, wurden Aluminiumschienen oben und unten an den Holzrahmen montiert. In der Schiene, welche am oberen Rand des Rahmens befestigt wird, wurden drei gleichmäßig verteilte Löcher mit einem Durchmesser von 3mm gebohrt. Nun wird ein Draht mit 1,5mm Durchmesser durch die Löcher gesteckt und eine Art Halbkreis geformt, sodass der Spiegel auf die in Abbildung 15 gezeigten Haken aufgehängt werden kann. |

1. **Ubuntu-Server Setup**

Der Server dient als Schnittstelle zwischen dem Raspberry Pi, welcher die grafische Benutzeroberfläche auf dem Display anzeigt, und einer Web- bzw. Smartphone-App. Auf dem Server ist die Datenbank und die Website abgelegt, sodass von überall darauf zugegriffen werden kann.

* 1. **Installation der benötigten Software**

Zu dem Server kann prinzipiell per SSH über den Port 22 zugegriffen werden. Bei den Betriebssystemen Linux bzw. MacOS ist keine zusätzliche Software für den SSH Zugriff notwendig, da dies mithilfe der Kommandozeile funktioniert. Bei Windows wird eine zusätzliche Anwendung wie z.B. Putty benötigt.

* 1. **Server Login**

Wird nun versucht per SSH auf den Server zuzugreifen, so wird die IP-Adresse des Servers benötigt. Unter Linux bzw. OSX kann der Zugriff mit folgendem Befehl erfolgen.

|  |
| --- |
| ssh 173.212.224.225 |

Codeabschnitt 1: Serverzugriff per SSH mit Linux/Mac-OS

Bei Windows sieht der Zugriff per SSH mithilfe von Putty folgendermaßen aus:

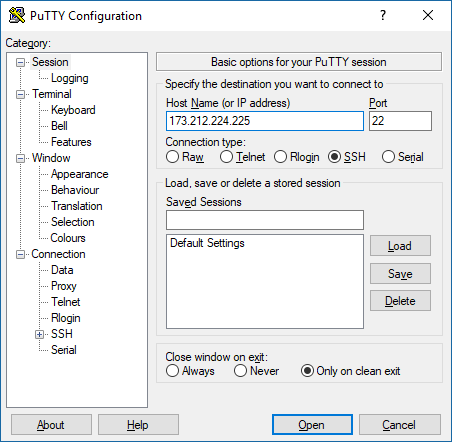


Abbildung 16:Zugriff auf Server mit Putty

Wenn der Verbindungsaufbau erfolgreich war, so sollte eine Benutzernamens- und Passwortabfrage erscheinen. In den meisten Fällen wird der Benutzername für den Server *root* und als Passwort anfangs nichts gespeichert sein. Nach erfolgreicher Eingabe sollte eine gewöhnliche UNIX-Shell geöffnet werden, welche den aktuellen Benutzernamen (in diesem Fall wurde ein eigener Benutzer *reflecton* angelegt) und anschließend nach dem „*@“* die IP-Adresse des Servers anzeigt. (Siehe Abbildung 17).

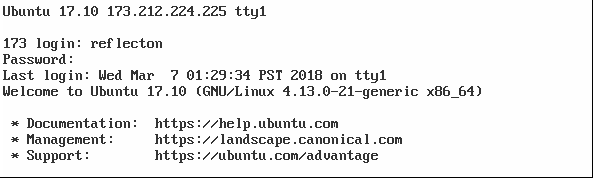


Abbildung 17: Bildschirm nach erfolgreichem Log In-Vorgang

* 1. **Aktualisierung des Servers**

Um sicherzustellen, dass die aktuellste Version der Pakete installiert ist, wird der Server aktualisiert. Dies funktioniert bei fast allen Linux-Versionen mithilfe der *apt-url*. Hierfür muss jedoch die Berechtigung zur Durchführung des Prozesses des *root*-Benutzers eingeholt werden, falls dieser nicht ohnehin bereits verwendet wird.

|  |
| --- |
| sudo apt-get update |

Codeabschnitt 2: Updatebefehl des Servers

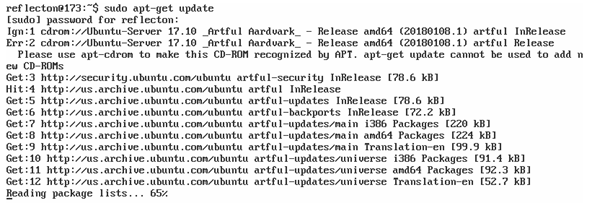


Abbildung 18: Updatevorgang des Servers

Nun werden die aktuellsten Versionen der installierten Pakete heruntergeladen. Zum Installieren der Updates muss jedoch ein separater Befehl eingeben werden, welcher wie folgt aussieht:

|  |
| --- |
| sudo apt-get upgrade |

Codeabschnitt 3: Installieren der Updates

Nun wurde der Server auf den aktuellsten Stand gebracht und es kann nun mit der Installation der Software, welche für die Funktion der Schnittstelle benötigt wird, begonnen werden.

* 1. **Installation des Apache-Servers**

Um nun den Server prinzipiell als Webserver nutzen zu können, wird das *Apache2* Paket benötigt. Diese werden wie bereits die Pakete davor mithilfe der *apt-url* installiert.

|  |
| --- |
| sudo apt-get install apache2 |

Codeabschnitt 4: Installation Apache-Pakete

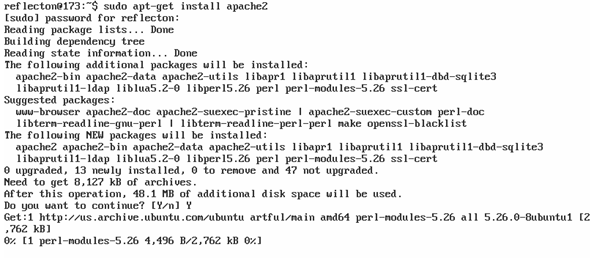
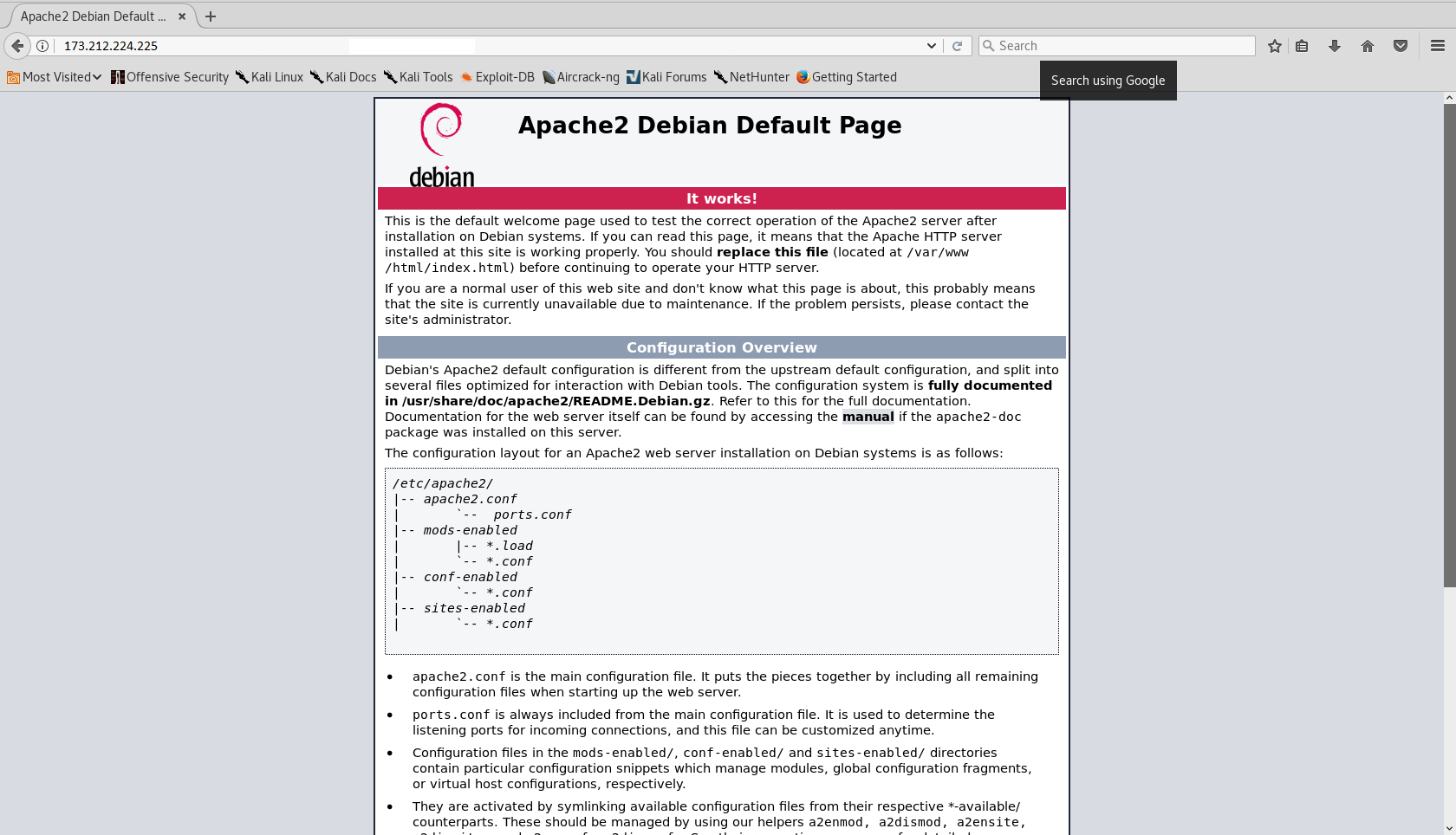


Abbildung 19: Installationsvorgang von Apache-Webserver Pakete

Ist die Installation des *Apache2* Paketes abgeschlossen, kann diese direkt getestet werden, indem in einem Browser die IP-Adresse des Servers eingetragen wird. Das Ergebnis sollte nun wie in Abbildung 20 gezeigt aussehen.

Abbildung 20: Testen der Apache Funktionalität

* 1. **Installation von MariaDB**

Um MariaDB auf dem Server installieren zu können, werden mehrere Komponenten benötigt. Einerseits die Pakete für den Server und auch die für den Client. Wenn die beiden Pakete installiert werden sieht der Befehl folgendermaßen aus.

|  |
| --- |
| sudo apt-get install mariadb-server mariadb-client |

Codeabschnitt 5: Installation von MariaDB am Server

Alternativ kann natürlich auch zu MySQL-Datenbank gewechselt werden, wofür ebenfalls das Server- als auch das Clientpaket benötigt wird. In diesem Fall würde es ein sehr ähnliches Ergebnis liefern, da MariaDB eine neuere Version von MySQL ist. Der Installationspfad sieht nun wie folgt aus:

|  |
| --- |
| sudo apt-get install mysql-server mysql-client |

Codeabschnitt 6: Installation alternativ zu MariaDB von MySQL

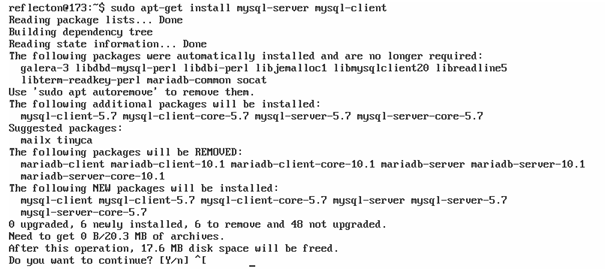


Abbildung 21: Installation der MySQL Pakete am Server

Sobald die Installation abgeschlossen ist, sollte direkt ein Root-Passwort für die Datenbank vergeben und ein paar Sicherheitseinstellungen durchgeführt werden. Das Passwort für die Datenbank ist in weiterer Folge für die Webanwendung von Bedeutung, da ansonsten kein Zugriff auf eine Tabelle oder ähnliches möglich ist. Bei den Sicherheitseinstellungen ist zu empfehlen, den *Remote-Root-Log-In* zu deaktivieren. Dies erhöht die Sicherheit für die Datenbank deutlich. Gelingt es einer unbefugten Person, die Zugangsdaten des *root*-Benutzers durch *Sniffing* auszuspionieren, kann sich dieser nicht über eine *UNIX*-Shell in die Datenbank einloggen und wichtige Datensätze modifizieren.

Eine weitere dringend zu empfehlende Sicherheitseinstellung betrifft das Löschen der Testdatenbank. Die Testdatenbank wird in den meisten Fällen nicht benötigt und stellt nur ein Sicherheitsproblem für den Server dar.

Zu guter Letzt werden noch die Zugriffsrechte der einzelnen Benutzer aktualisiert um die eben durchgeführten Änderungen wirksam zu machen. Die Einstellungen können durch folgenden Befehl erneut geändert werden:

|  |
| --- |
| sudo mysql\_secure\_installation |

Codeabschnitt 7: Installation der Sicherheitsoptionen für MySQL/MariaDB

Mit folgendem Befehl kann mithilfe der Kommandozeile als *root*-Benutzer auf die Datenbank zugegriffen werden.

|  |
| --- |
| sudo mysql -u root -p “Password des root users” |

Codeabschnitt 8: Zugriff auf MySQL-Datenbank mit Terminal

Die Verbindung zur Datenbank kann mithilfe des darunter angeführten Befehls beendet werden.

|  |
| --- |
| exit |

Codeabschnitt 9: Schließen der Datenbankverbindung über das Terminal

* 1. **Installation von PHP-7**

Wie bereits davor mehrmals erwähnt, werden auch die Pakete für *PHP7* mithilfe der *apt-url* installiert. Zusätzlich ist eine Bibliothek für Apache zum Ausführen von PHP-Anwendung auf dem Server notwendig.

|  |
| --- |
| sudo apt-get -y install php7.0 libapache2-mod-php7.0 |

Codeabschnitt 10: Installation von PHP7.0-Paktete

Nach erfolgreichem Abschluss der Installation wird der *Apache-Service* neu gestartet, wodurch die neu installierten Pakete installiert werden. Dies wird mit dem anschließenden Befehl erzielt.

|  |
| --- |
| sudo systemctl restart apache2 |

Codeabschnitt 11: Neustarten des Webservers

Jetzt können die aktuellen Fortschritte zum ersten Mal getestet werden, ob PHP 7 auch installiert und funktionstüchtig ist. Hierfür wird eine *test.php*-Datei erstellt, welche die aktuellen Einstellungen zu der Programmiersprache anzeigt. Dafür wird zuerst mithilfe des nachfolgenden Befehles in den *Apache-Web*-Ordner gewechselt, da von diesem aus PHP-Skripte übers Internet erreicht werden können.

|  |
| --- |
| cd /var/www/html |

Codeabschnitt 12: Wechsel in das Root-Verzeichnis des Webservers

Um das PHP-Skript zu erstellen, wird der anschließende Befehl eingegeben:

|  |
| --- |
| nano test.php |

Codeabschnitt 13: Erstellung einer Testdatei zum Testen des Servers

Nun öffnet sich in der aktuellen Shell ein Editor, in die der nachfolgende Code eingefügt wird:

|  |
| --- |
| <?php phpinfo(); ?> |

Codeabschnitt 14: Code (test.php)

Mit der Tastenkombination *Strg. + X* wird der Editor geschlossen und das Speichern der aktuellen Datei wird mit *Y* bestätigt.

Damit die aktuellen PHP-Einstellungen, welche in Abbildung 21 zu sehen sind im Browser angezeigt werden, wird nach der IP-Adresse des Servers der Zusatz */test.php* eingefügt.

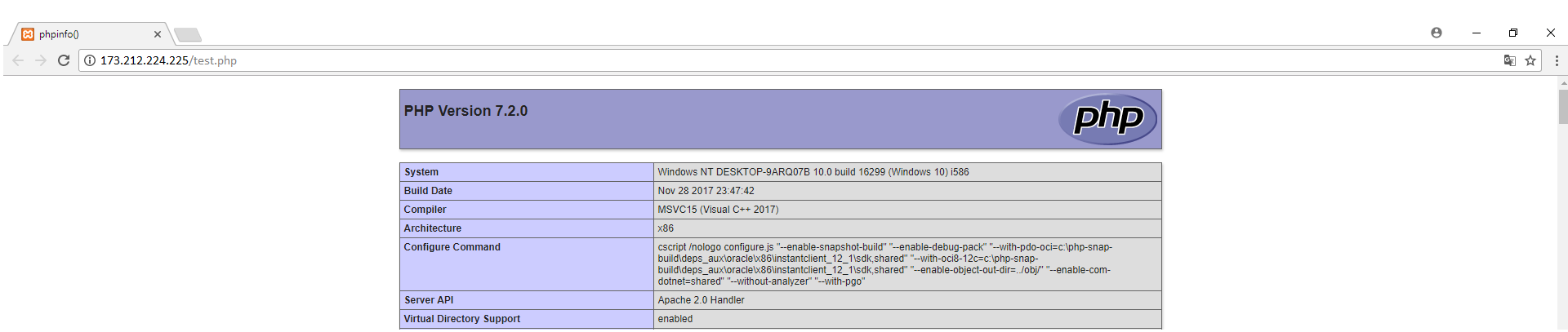


Abbildung 22: Ausgabe *(*test.php*)*

Damit die Kommunikation zwischen PHP und MySQL bzw. MariaDB auch funktioniert, wird ein zusätzliches Paket für PHP benötigt, welches mit folgendem Befehl installiert werden kann.

|  |
| --- |
| sudo apt-get -y install php7.0-mysql |

Codeabschnitt 15: Installation der MySQL Erweiterungen für PHP

Nach Abschluss des Installationsvorganges wird der Apache-Webserver neu gestartet um die letzten Änderungen gültig zu machen und anschließend ist die Konfiguration eines Webservers abgeschlossen.

|  |
| --- |
| sudo systemctl restart apache2 |

Codeabschnitt 16: Neustarten des Apache-Webservers

1. **MySQL-Datenbank**

Um die Benutzerdaten an einem gemeinsamen Ort zu speichern, wird eine Datenbank in MySQL erstellt. Diese wird anschießend auf den Server geladen, da diese die Schnittstelle zwischen dem Raspberry Pi und den Applikationen z.B.: Smartphone-App und Website darstellt.

* 1. **Verwaltung der MySQL Datenbank**

Um für den Zugriff auf die Datenbank nicht den Root-Benutzer verwenden zu müssen, wird ein neuer Benutzer mit den angepassten Zugriffsrechten erstellt. Die nachfolgenden Befehle werden nun über das Terminal direkt eingeben.

* + 1. Neuen Benutzer in der Datenbank erstellen

Der neue Benutzer kann entweder über das Terminal per SSH-Zugriff erstellt werden oder über das *PHPMyAdmin*-Webinterface, welches über den Browser erreicht werden kann. Der Benutzer wird mit dem *CREATE USER*-Statement in MySQL erstellt. Anschließend wird ein Benutzername festgelegt. Nach der *IDENTIFIED BY*-Anweisung wird ein Passwort übergeben.

|  |
| --- |
| CREATE USER ‘java‘@‘localhost‘ IDENTIFIED BY ‘password‘; |

Codeabschnitt 17: Benutzer erstellen

Mit dem Befehl *GRANT ALL PRIVILIGES* werden die Zugriffsrechte für die Datenbank erteilt. Anschließend wird die entsprechende Datenbank ausgewählt. In diesem Fall werden die vollen Zugriffsrechte auf allen Datenbanken an den „root“-Benutzer erteilt. Weiter im Code wird nach dem *INDENTIFIED BY*-Befehl das Passwort des jeweiligen Benutzers eingefügt. Der Befehl *WTH GRANT OPTION* sorgt dafür, dass auch dieser Benutzer Zugriffsrechte verteilen kann. Um die Änderungen in die Datenbank zu schreiben, werden diese mit *FLUSH PRIVILIGES* neu geladen und in der Datenbank gültig.

|  |
| --- |
| GRANT ALL PRIVILIGES ON \*.\* TO ‘root‘@‘%‘ INDENTIFIED BY ‘Password‘   WITH GRANT OPTION;  FLUSH PRIVILIGES |

Codeabschnitt 18: Verteilen von Zugriffsrechten

* + 1. Erstellung der Datenbank in PHPMyAdmin

Um mit der *PHPMyAdmin*-Oberfläche arbeiten zu können, muss diese auf dem Server installiert sein. Falls dies nicht der Fall ist kann diese mit folgendem Befehl installiert werden.

|  |
| --- |
| sudo apt-get install phpMyAdmin |

Codeabschnitt 19: PhpMyAdmin installieren

Nun kann auf die Datenbanken mittels der *PHPMyAdmin*-Oberfläche zugegriffen werden. Dies funktioniert durch Eingeben der IP-Adresse des Servers und dem Zusatz */phpmyadmin*, welcher nach der IP-Adresse angehängt wird. Die nachfolgende Abbildung zeigt, wie das Ergebnis aussehen sollte, wenn die *PHPMyAdmin*-Oberfläche erfolgreich installiert wurde.

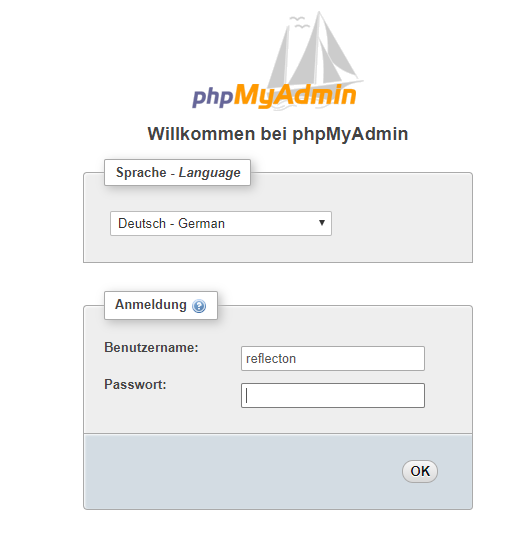


Abbildung 23: PHPMyAdmin Oberfläche nach erfolgreicher Installation

Sobald der Log In-Vorgang erfolgreich durchgeführt werden konnte, können alle Datenbanken verwaltet werden bzw. Neue hinzugefügt werden, was mithilfe des *Neu*-Labels realisiert werden kann.

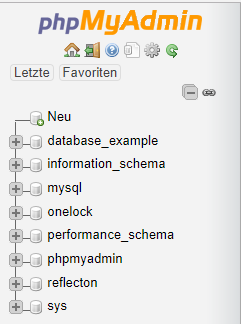


Abbildung 24: Verwaltung der bereits bestehenden Datenbanken

Im nächsten Schritt wird nun eine neue Datenbank mit dem Namen *reflecton* angelegt. Hierbei wird auf das *Neu*-Label geklickt und anschließend ein Name für die Datenbank (in unserem Fall *reflecton*) eingegeben. Nun wird, wie die Abbildung 25 zeigt, in der eben angelegten Datenbank eine neue Tabelle erzeugt.

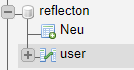


Abbildung 25: Erzeugen einer neuen Tabelle in der Datenbank

Jedem neu hinzugefügten Benutzer wird eine ID zugewiesen, welche automatisch erhöht wird, wenn ein weiterer hinzugefügt wird. Diese dient der Orientierung der Anwendung am Spiegel, damit auch wirklich die Anordnung des jeweiligen Benutzers und nicht die eines anderen übernommen wird. Der Primärschlüssels wird, wie in Abbildung 26 ersichtlich, mit einem kleinen goldenen Schlüssel neben dem Namen der Spalte, angezeigt.



Abbildung 26: ID-Feld in der Tabelle

Die nächsten Felder dienen der zusätzlichen und eindeutigeren Identifizierung des jeweiligen Benutzers mithilfe des Vor- bzw. Nachnamens und einer E-Mail-Adresse. Logischerweise wird auch jeder Benutzer mit einem Passwort geschützt, damit die persönlichen Daten eines jeden Benutzers geschützt sind. Das Passwort wird als *Hash* in der Passwortspalte abgespeichert. Die Definition der vier Felder ist in Abbildung 27 ersichtlich.

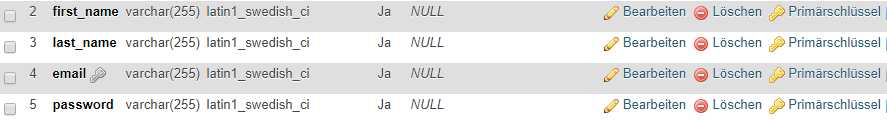


Abbildung 27: Zusatzinformationen über den Benutzer

In der letzten Spalte ist sowohl die Anordnung des Spiegels, als auch die Zugangsdaten für die E-Mail- und Kalenderanwendung gespeichert. Die Anordnung der Anwendung wird in XML-Style durchgeführt. Dies bedeutet, dass jedes Widget mit dem *<widget>-Tag* geöffnet und mithilfe des *</widget>-Tag* geschlossen wird. In dem *Tag* der E-Mail werden zusätzlich durch einen Doppelpunkt getrennt die Zugangsdaten für das E-Mail Benutzerkonto angegeben. Der vordere Teil symbolisiert die E-Mail-Adresse, der dahinter befindliche das zugehörige Passwort, mit welchem das Benutzerkonto geschützt ist.

|  |
| --- |
| <widget>email:m.mustermann@mail.com:abc</widget>  <widget>calendar:m.mustermann@icloud.com:abc</widget>  <widget>weather</widget>  <widget>clock</widget>" |

Codeabschnitt 20: Definition der Widgets

1. **PHP-Interface zur Datenbank**

Um den Reflecton-Spiegel per Smartphone-App bedienen zu können, sind zwei Schnittstellen notwendig. Zum einen ist es die Schnittstelle zwischen Smartphone und Datenbank, zum anderen die Schnittstelle zwischen Datenbank und Reflecton.

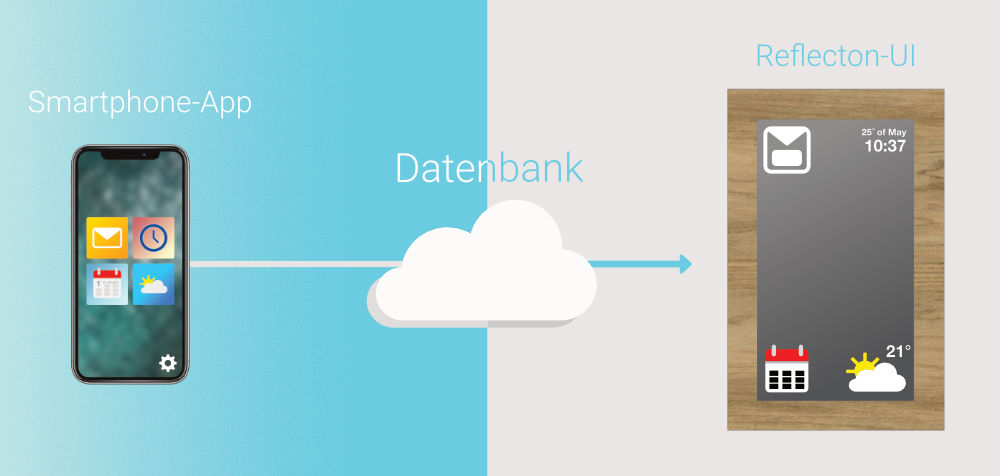


Abbildung 28: Schnittstellen zwischen den Komponenten

In diesem Kapitel wird die Verbindung zwischen App und Datenbank behandelt.

* 1. **Warum PHP?**

Um einem Client von außen den Zugriff zur Datenbank zu gewähren, wird ein PHP-Skript oder eine Library wie z.B. JDBC für Java benötigt. Da es eine iOS- und eine Android-App geben soll, ist eine möglichst universelle Lösung notwendig. Für eine universelle Lösung ist ein PHP-Skript geeignet, da es von iOS und von Android aufgerufen werden kann. Hingegen bei der Schnittstelle von Datenbank zu Reflecton-User-Interface eignet sich JDBC, da auf dieser Seite des Projektes nur in Java programmiert werden muss.

* 1. **Logik**

Das PHP-Interface wurde mit vielseitiger Anwendung von Objektorientierung programmiert. Außerdem wurde ein großer Wert auf vielseitige Einsetzbarkeit gelegt. Es ist nun möglich, dass das PHP-Interface nahezu unverändert in einem anderen Projekt wiederverwendet werden kann. Dies wurde durch den Einsatz von sogenannten Config-Files ermöglicht. In diesen Konfigurationsdateien wird die Struktur der jeweiligen Datenbanktabelle sowie sonstige variable Parameter abgebildet.

* 1. **Struktur**

Im unten dargestellten Bild ist die Struktur des PHP-Interfaces ersichtlich. Das PHP-Interface besteht aus zwei Ordnern. Zum einen aus dem *api*-Ordner und zum anderen aus dem *includes*-Ordner.

|  |  |
| --- | --- |
| Abbildung 29: Struktur | Im *api*-Ordner befinden sich vier PHP-Dateien mit dem Interface-Präfix. Diese Dateien sind nicht mit der echten Schnittstelle zu verwechseln. Es handelt sich hierbei um ein grafisches HTML-UI. Die Schnittstelle zum Smartphone sind *setdata.php*, *signin.php* und *signup.php*. |
| Im *includes*-Ordner befinden sich die Klassen *Column.php*, sowie *Database.php*. Der wichtigere Part von beiden ist *Database.php*. Diese Datei regelt alle möglichen Zugriffe auf die Datenbank. Dabei werden in unterschiedlichen Funktionen SQL-Statements zusammengebaut und ausgeführt. Außerdem werden mit den Rückgabewerten Informationen an die Smartphone-App geliefert. |

* 1. **Konfigurationsdateien**

Die folgenden Unterkapitel beschreiben die Dateien, die im vorherigen Kapitel angesprochen wurden. Es wird die Logik des Programmes genauer erklärt und wie die PHP-Schnittstelle mit der Smartphone-App zusammenspielt.

Wie zuletzt erklärt, ist das gesamte Skript Config-File gesteuert. Von jenen gibt es insgesamt vier im *includes*-Ordner. Der Sinn dieser nicht statischen Programmiermethode ist es, das Programm spielendleicht an die verschiedensten Tabellen in Datenbanken anpassen zu können. Die folgenden Konfigurationsdateien werden zur konkreten Ausführung von Reflecton benötigt.

* + 1. Reflecton/includes/config.php

|  |
| --- |
| **<?php****const *DB\_HOST*** = **'173.212.224.225'**; **const *DB\_USER\_NAME*** = **'reflecton'**; **const *DB\_USER\_PWD*** = **'\*\*censored\*\*'**; **const *DB\_NAME*** = **'reflecton'**; **const *TABLE\_NAME*** = **'user'**; |

Codeabschnitt 21: Allgemeine Konfigurationsdatei (config.php)

Die Konfigurationsdatei *config.php* stellt die Datenbankserver-Einstellungen dar. Es muss die IP-Adresse des zugehörigen Servers, der Benutzername der SQL-Datenbank mit zugehörigem Passwort, der Name der Datenbank, sowie der Name der Tabelle definiert werden.

* + 1. Reflecton/includes/signup\_config.php

|  |
| --- |
| **<?php****require '../includes/Column.php'**; $columnElementsSignUp = **array**(  **new** Column(**'first\_name'**, **'Firstname'**, **'varchar(255)'**, **null**),  **new** Column(**'last\_name'**, **'Lastname'**, **'varchar(255)'**, **null**),  **new** Column(**'email'**, **'E-Mail'**, **'varchar(255)'**, **'UNIQUE(email)'**),  **new** Column(**'password'**, **'Password'**,**'varchar(255)'**, **null**),  **new** Column(**'data'**, **'Data'**, **'varchar(255)'**, **null**) );$rowElements = **array**(); |

Codeabschnitt 22: Konfigurationsdatei für die Kontoerstellung (signup\_config.php)

Da beim Erstellen eines Kontos alle möglichen Parameter benötigt werden, enthält das Array *$columnElementsSignUp*, welches in der Datei *signup\_config.php* definiert ist, alle Spalten aus der Datenbank. Die *user*-Tabelle ist somit ebengleich wie dieses Array.

* + 1. Reflecton/includes/signin\_config.php

|  |
| --- |
| **<?php****require '../includes/Column.php'**; $columnElementsSignIn = **array**(  **new** Column(**'email'**, **'E-Mail'**, **'varchar(255)'**, **'UNIQUE(email)'**),  **new** Column(**'password'**, **'Password'**,**'varchar(255)'**, **null**) );$rowElements = **array**(); |

Codeabschnitt 23: Konfigurationsdatei für die Anmeldung (signin\_config.php)

Da beim Einloggen nur E-Mail und Passwort benötigt werden, enthält das Array *$columnElementsSignIn*, welches in der Datei *signin\_config.php* definiert ist, nur zwei Elemente. Diese Elemente sind äquivalent zu den Spalten *email* und *password* in der Datenbank.

* + 1. Reflecton/includes/setdata\_config.php

|  |
| --- |
| **<?php****require '../includes/Column.php'**; $columnElementsSetData = **array**(  **new** Column(**'email'**, **'E-Mail'**, **'varchar(255)'**, **'UNIQUE(email)'**),  **new** Column(**'data'**, **'Data'**, **'varchar(255)'**, **null**) ); $rowElements = **array**(); |

Codeabschnitt 24: Konfigurationsdatei für Datenaktualisierungen (setdata\_config.php)

Später muss beim Verschieben eines Widgets in der iOS-App das Feld *data* aktualisiert werden. Um jenes Feld ändern zu können, werden zwei Elemente im Array *$columnElementsSetData*, welches in der Datei *setdata\_config.php* definiert ist, benötigt. Diese Elemente sind *email*, welches als Schlüssel zum Datensatz dient, sowie *data*, welches die neue Anordnung der Widgets beinhaltet.

* + 1. Reflecton/includes/getdata\_config.php

|  |
| --- |
| **<?php require '../includes/Column.php'**; $columnElementsGetData = **array**(  **new** Column(**'email'**, **'E-Mail'**, **'varchar(255)'**, **'UNIQUE(email)'**) ); $rowElements = **array**();$resultField = **'data'**; |

Codeabschnitt 25: Konfigurationsdatei für das Laden von Daten (getdata\_config.php)

Um im Endprodukt bei jedem Neustart der iOS-App die derzeitige Anordnung der Widgets zu laden, wird nur das Element *email* im Array *$columnElementsGetData*, welches in der Datei *getdata\_config.php* definiert ist, benötigt.

* 1. **Klassen**

Im *includes*-Ordner befinden sich die Klassen *Column*, sowie *Database*. Im Gegensatz zur Klasse *Column*, welche nur eine Hilfsklasse zur Speicherung von Tabellenstrukturen darstellt, ist die Klasse *Database* das wahrscheinlich wichtigste Glied des Skriptes. In Fachsprache ausgedrückt dient die *Database*-Klasse zur generellen Manipulation der Datenbank.

* + 1. Reflecton/includes/Column.php

|  |
| --- |
| **<?php****class** Column {  **private $name**;  **private $shownName**;  **private $type**;  **private $other**;   **public function** \_\_construct($name, $shownName, $type, $other) {  $this->**name** = $name;  $this->**shownName** = $shownName;  $this->**type** = $type;  $this->**other** = $other;  }  **public function** getName() { **return** $this->**name**; }  **public function** getShownName() { **return** $this->**shownName**; }  **public function** getType() { **return** $this->**type**; }  **public function** getOther() { **return** $this->**other**; }  **public function** toString() {  $returnValue = **''**;  **if**($this->**other** != **null**) {  $returnValue = $this->**name** . **' '** . $this->**type** . **', '** . $this->**other**;  }  **else** {  $returnValue = $this->**name** . **' '** . $this->**type**;  }  **return** $returnValue;  } } |

Codeabschnitt 26: Klasse für Darstellung der Tabelle (Column.php)

Die Klasse *Column* ist nur eine Hilfsklasse, welche das Schema einer Tabelle abbildet. In der Objektorientierung spricht man hierbei von der Implementierung eines neuen Datentyps. Das Wort Datentyp trifft es in diesem Fall sehr gut, da die Klasse nur zur Speicherung einer Tabellenstruktur dient. Dies lässt sich auch aus den *Config*-Files unschwer erkennen, da diverse Instanzen dieser spezifischen Klasse implementiert wurden. In dieser Klasse sind die zwei einzigen Funktionen sogenannte „Getter“-Methoden sowie „Setter“-Methoden.

Membervariablen

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| $name | Name des Feldes in der Datenbank z.B. 'email' |
| $shownName | Name der im HTML-Formular angezeigt werden soll z.B. 'E-Mail' |
| $type | Datentyp des Feldes z.B. 'varchar(255)' |
| $other | Spezialspezifikation z.B. 'UNIQUE(email)' |

* + 1. Reflecton/includes/Database.php

Die Klasse Database ist zur Interaktion und Manipulation mit der Datenbank vorhanden. Sie ist somit das zentrale Element des Skripts. Neben zahlreichen Membervariablen sind im Skript auch einige Funktionen implementiert, welche alle unterschiedliche Manipulationsvorgänge an der Datenbank vornehmen. Diese Funktionen und Membervariablen werden unten einzeln erklärt, da der Code zu groß und somit zu unhandlich wäre.

Membervariablen

Wie im Programmcode ersichtlich, werden die Membervariablen über den Konstruktor initialisiert. Diese Daten kommen ursprünglich aus der Hauptkonfigurationsdatei *config.php*, oder von den aufgabenspezifischen Konfigurationsdateien wie *getdata\_config.php*, *setdata\_config.php*, *signin\_config.php*, sowie *signup\_config.php*.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| $host | Unter dieser Membervariable wird eine Zeichenkette verstanden, welche den Host darstellt. Dies kann beispielsweise die IP-Adresse '173.212.224.225', oder auch der String 'localhost' sein. |
| $userName | Der Benutzername ist zur Anmeldung an einer SQL-Datenbank notwendig. Durch ihn ist eine Absicherung der Datenbank gegeben. Außerdem ist jedem Benutzer eine gewisse Anzahl an Rechten zugeschrieben. So ist es z.B. nicht schlau den Benutzer 'root' zu verwenden, da im PHP-Skript mit diesem alles gemacht werden kann. |
| $userPassword | Um eine erfolgreiche Authentifizierung am Datenbankserver zu erzielen, wird zugehörig zum Benutzernamen selbstverständlich auch ein Passwort benötigt. |
| $databaseName | Diese Membervariable enthält den Namen der Datenbank am Server. |
| $tableName | Der Name der Tabelle wird in dieser Membervariable gespeichert. |
| $columnElements | Im Gegensatz zu den vorherigen Membervariablen, welche ihren Ursprung alle im File *config.php* finden, erhält speziell diese Membervariable in jeder aufgabenspezifischen Datei, wie beispielsweise *getdata\_config.php*, ihre Erstinitialisierung. |
| $databaseConnection | Um eine Verbindung zur Datenbank aufzubauen und jene auch noch zu steuern, wird diese Membervariable unumgänglich benötigt. Die Initialisierung dieser Variable erfolgt durch die Verbindungsanfrage in der Funktion *connectToDatabase()*. |

Funktionen

Neben den oben genannten Membervariablen gibt es noch einen Konstruktor sowie zahlreiche weitere Funktionen, welche gesamt das Herzstück unseres Programmes darstellen. Diese Funktionen sind verantwortlich für Manipulationsvorgänge und diverse andere Tätigkeiten.

Der Konstruktor *\_\_construct(...)* hat die Aufgabe, einen Großteil der Membervariablen zu initialisieren.

|  |
| --- |
| **public function** \_\_construct($host, $userName, $userPassword, $databaseName, $tableName, $columnElements) {$this->**host** = $host;  $this->**userName** = $userName;  $this->**userPassword** = $userPassword;  $this->**databaseName** = $databaseName;  $this->**tableName** = $tableName;  $this->**columnElements** = $columnElements; } |

Codeabschnitt 27: Konstruktor (Database.php)

Wie im obigen Programmcode ersichtlich, werden die übergebenen Parameter den Membervariablen der individuellen Instanz übergeben. Besonders hervorzuheben ist die Konvention für einen Konstruktor *\_\_construct(...)* in PHP.

Um weitere Funktionen ausführen zu können, wird zuerst ein Verbindungsaufbau zur Datenbank benötigt. Dafür ist die Funktion *connectToDatabase()* zuständig.

|  |
| --- |
| **public function** connectToDatabase() {$this->**databaseConnection** = **new** mysqli($this->**host**, $this->**userName**,  $this->**userPassword**, $this->**databaseName**);**if** (*mysqli\_connect\_errno*()) {  **echo "Failed to connect to MySQL: "** . *mysqli\_connect\_error*();  } } |

Codeabschnitt 28: Funktion zum Verbinden der Datenbank (Database.php)

In der ersten wirksamen Zeile wird ein neues Objekt vom Typ *mysqli* erzeugt und der Membervariable *databaseConnection* zugewiesen. Übergeben werden diverse Parameter, welche zum Auffinden der Datenbank benötigt werden. Das *databaseConnection*-Objekt wird in anderen Funktionen benötigt, da über die Verbindung weitere Funktionen aufgerufen werden können. In der if-Anweisung wird überprüft, ob ein Fehler beim Verbindungsaufruf geschehen ist. Sollte ein Fehler aufgetreten sein, so wird dieser mithilfe eines Strings ausgegeben.

Eine Verbindung soll im Regelfall auch wieder geschlossen werden. Dieser Auftrag wird der Funktion *closeConnectionToDatabase()* erteilt.

|  |
| --- |
| **public function** closeConnectionToDatabase() {$this->**databaseConnection**->close(); } |

Codeabschnitt 29: Funktion zum Schließen der Verbindung (Database.php)

Die Funktion ruft eigentlich nur eine Schließfunktion des *databaseConnection*-Objekts auf. Ein einfacher Aufruf der Schließfunktion würde anstatt unserer eigenen Funktion *closeConnectionToDatabase()* auch in jedem Fall reichen. Dennoch stellt die eigene Funktion einen Formschluss im Programmierstil sicher.

Neben den vorhin erläuterten Verbindungsfunktionen werden für die Datenbank auch noch Manipulationsfunktionen benötigt. Wie der Name der Funktion *createTable()* schon verrät, wird hierbei eine Tabelle in der Datenbank erstellt, sofern diese noch nicht existiert. Bei der Erstellung eines Kontos (Sign Up) muss logischerweise vorher sichergestellt werden, dass die gewünschte Tabelle auch existiert.

|  |
| --- |
| **public function** createTable() {$sqlQuery = **"CREATE TABLE IF NOT EXISTS „** . $this->**tableName** .**" (id int NOT NULL AUTO\_INCREMENT, PRIMARY KEY(id)"**;   **for** ($i = 0; $i < *count*($this->**columnElements**); $i++) {  $sqlQuery = $sqlQuery . **","** . $this->**columnElements**[$i]->toString();  }  $sqlQuery = $sqlQuery . **")"**;$stmt = $this->**databaseConnection**->prepare($sqlQuery);  $stmt->execute();  $error = $stmt->**error**;  $stmt->close(); **return** $error; } |

Codeabschnitt 30: Funktion zum Erstellen einer Tabelle (Database.php)

Im ersten Teil des Blocks wird ein Query in Form eines Strings mithilfe einer Schleife zusammengebaut. Dabei wird auf verschiedenste Membervariablen zugegriffen und deren Werte werden sinnvoll integriert. Die Variable *$sqlQuery* sieht in schlussendlicher Form wie folgt aus:

|  |
| --- |
| **CREATE TABLE** IF **NOT EXISTS user** (id **int NOT NULL** AUTO\_INCREMENT, **PRIMARY KEY**(id), first\_name **varchar**(255), last\_name **varchar**(255), email **varchar**(255), **UNIQUE**(email), password **varchar**(255), **data varchar**(255)) |

Codeabschnitt 31: SQL-Statement zum Tabelle erstellen (Database.php)

Nachdem der Query zusammengestellt wurde, wird dieser mit *prepare(...)* vorbereitet und mit *execute()* ausgeführt. Danach wird die Ausführung gestoppt und ein möglicher Fehler zurückgegeben. Wenn kein Fehler aufgetreten ist, so kann nun davon ausgegangen werden, dass eine Tabelle vorhanden ist.

Die Funktion *selectRow()* dient zum Auswählen eines bestimmten Datensatzes. Diese Funktion wird zum Beispiel beim Log In (Sign In) benötigt.

|  |
| --- |
| **public function** selectRow($rowElements) {$sqlQuery = **"SELECT** *\** **FROM "** . $this->**tableName** . **" WHERE "**;  **for** ($i = 0; $i < *count*($this->**columnElements**); $i++) {  $sqlQuery = $sqlQuery . $this->**columnElements**[$i]->getName();  $sqlQuery = $sqlQuery . **"="** . **"'"** . $rowElements[$i] . **"'"**;  **if**($i < *count*($this->**columnElements**) - 1)  {  $sqlQuery = $sqlQuery . **" AND "**;  }  }$result = $this->**databaseConnection**->query($sqlQuery);$row = **""**; **while**(($array = *mysqli\_fetch\_array*($result)) != **null**) {  *ob\_start*();  *var\_dump*($array);  $row = *ob\_get\_clean*();  }$message = 0;**if**(*strlen*($row) == 0) {$message = 0;  }  **else** {$message = 1;  }**return** $message; } |

Codeabschnitt 32: Funktion zum Selektieren eines Datensatzes (Database.php)

Im ersten Drittel des Codes wird wieder ein Query zusammengebaut. Dieser schaut bei der Anmeldung des Benutzers Max Mustermann (E-Mail: "m.mustermann@mail.com", Passwort: "abc") wie folgt aus. Die Parameter wie E-Mail und Passwort müssen der Funktion über die Variable *$rowElements* übergeben werden.

|  |
| --- |
| **SELECT** *\** **FROM user WHERE** email=**'m.mustermann@mail.com' AND** password=**'abc'** |

Codeabschnitt 33: SQL-Statement zum Selektieren eines Benutzers (Database.php)

Dieser Query liefert mit der Funktion *query(...)* einen einzigen Datensatz, sofern dieser auch existiert. Die Bedingungen dafür sind in der WHERE-Klausel zu finden.

Innerhalb der while-Schleife wird das Array in einen String *$row* umgewandelt. Dieser String sieht dann wie folgt aus:

|  |
| --- |
| array(12) {   [0]=> string(1) **"2"** ["id"]=> string(1) **"2"**  [1]=> string(3) **"Max"** ["first\_name"]=> string(3) **"Max"**  [2]=> string(10) **"Mustermann"** ["last\_name"]=> string(10) **"Mustermann"**  [3]=> string(21) **"m.mustermann@mail.com"** ["email"]=> string(21) **"m.mustermann@..."**   [4]=> string(3) **"abc"** ["password"]=> string(3) **"abc"**  [5]=> string(1) **"<widget>email..."** ["data"]=> string(1) **"<widget>email..."** } |

Codeabschnitt 34: Datensatz eines Benutzers (Database.php)

Nach der while-Schleife folgen if-Anweisungen, welche überprüfen, ob der String *$row* einen Inhalt hat. Ist *$row* mit einem Wert wie z.B. mit dem in der obigen Box belegt, so wird "1" zurückgegeben. Dieser Einser wird von der Smartphone-App erhalten, sofern ein existierendes Konto beim Einloggen (Sign-In) gefunden wurde. Wird "0" zurückgegeben, so muss in der App eine Fehlermeldung angezeigt werden, da kein passendes Konto in der Datenbank gefunden wurde.

Im Unterschied zur Letzteren gibt die Funktion *selectRowReturnResult()* ein Array zurück, in welchem genau ein Datensatz abgespeichert ist. Bis auf das Weglassen des unteren Teiles der zuletzt besprochenen Funktion *selectRow()* sind beide Funktionen ebengleich.

|  |
| --- |
| **public function** selectRowReturnResult($rowElements) {$sqlQuery = **"SELECT** *\** **FROM "** . $this->**tableName** . **" WHERE "**;  **for** ($i = 0; $i < *count*($this->**columnElements**); $i++) {  $sqlQuery = $sqlQuery . $this->**columnElements**[$i]->getName();  $sqlQuery = $sqlQuery . **"="** . **"'"** . $rowElements[$i] . **"'"**;  **if**($i < *count*($this->**columnElements**) - 1) {  $sqlQuery = $sqlQuery . **" AND "**;  }  }$result = $this->**databaseConnection**->query($sqlQuery);$array = *mysqli\_fetch\_array*($result);**return** $array; } |

Codeabschnitt 35: Funktion zum Selektieren eines Datensatzes (Database.php)

Dieses Array wird mithilfe der Funktion *mysqli\_fetch\_array(...)* erstellt und über den Return-Wert zurückgegeben.

Die Funktion *updateRowSearchByFirstElement()* dient zum Aktualisieren eines bestimmten Werts. Dabei ist das erste Element des Arrays *$rowElements* das Element nachdem gesucht wird. Die weiteren Elemente sind der Ersatz für die bestehenden Elemente. Diese Methodik ist speziell beim Aktualisieren der aktuellen Widget-Anordnung in der Smartphone-App von Bedeutung.

|  |
| --- |
| **public function** updateRowSearchByFirstElement($rowElements) {$sqlQuery = **"UPDATE "** . $this->**tableName** . **" SET "**;  **for** ($i = 1; $i < *count*($this->**columnElements**); $i++) {  $sqlQuery = $sqlQuery . $this->**columnElements**[$i]->getName();  $sqlQuery = $sqlQuery . **"="** . **"'"** . $rowElements[$i] . **"'"**;  **if**($i < *count*($this->**columnElements**) - 1) {  $sqlQuery = $sqlQuery . **", "**;  }  }  $sqlQuery = $sqlQuery . **" WHERE "**;  $sqlQuery = $sqlQuery . $this->**columnElements**[0]->getName();  $sqlQuery = $sqlQuery . **"="** . **"'"** . $rowElements[0] . **"'"**;$result = $this->**databaseConnection**->query($sqlQuery);**return** $result; } |

Codeabschnitt 36: Funktion zum Aktualisieren eines Datensatzes (Database.php)

Innerhalb der for-Schleife wird der SQL-Query zusammengebaut. Dabei ist zu beachten, dass die Schleife ab dem Element 1 zu zählen beginnt. Dies hat zur Folge, dass alle Elemente größer oder gleich 1 aktualisiert werden. Das Element 0 dient als Schlüssel in der WHERE-Klausel. Im Programmcode lässt sich dies nach der for-Schleife finden.

|  |
| --- |
| **UPDATE user SET data**=**'<widget>email:m.mustermann@mail.com:pw1</widget>  <widget>calendar:m.mustermann@mail.com:pw1</widget>  <widget>weather</widget>  <widget>clock</widget>' WHERE** email=**'m.mustermann@mail.com'** |

Codeabschnitt 37: SQL-Statement zum Aktualisieren eines Datensatzes (Database.php)

Nachdem der Query fertiggebaut wurde, wird dieser mittels der Funktion *query(...)* ausgeführt und das Ergebnis *$result* zurückgegeben.

Die letzte Manipulationsfunktion *insertRow()* ist für das Einfügen eines neuen Datensatzes verantwortlich. Diese wird vor allem beim Erstellen eines Kontos (Sign Up) benötigt.

|  |
| --- |
| **public function** insertRow($rowElements) {$sqlQuery = **"INSERT INTO "** . $this->**tableName** . **" ("**;  **for** ($i = 0; $i < *count*($this->**columnElements**); $i++) {  $sqlQuery = $sqlQuery . $this->**columnElements**[$i]->getName();  **if**($i < *count*($this->**columnElements**) - 1) {  $sqlQuery = $sqlQuery . **", "**;  }  }  $sqlQuery = $sqlQuery . **") VALUES ("**;   **for** ($i = 0; $i < *count*($rowElements); $i++) {  $sqlQuery = $sqlQuery . **"'"** . $rowElements[$i] . **"'"**;  **if**($i < *count*($rowElements) - 1) {  $sqlQuery = $sqlQuery . **", "**;  }  }  $sqlQuery = $sqlQuery . **")"**;$stmt = $this->**databaseConnection**->prepare($sqlQuery);  $stmt->execute();  $error = $stmt->**error**;  $stmt->close();**if**(*strlen*($error) == 0) {$message = 1;  }  **else** {$message = 0;  }**return** $message; } |

Codeabschnitt 38: Funktion zum Einfügen eines Benutzers (Database.php)

In der ersten Hälfte des Codes befinden sich zwei for-Schleifen. Diese sind für das Zusammenbauen des Querys verantwortlich.

|  |
| --- |
| **INSERT INTO user** (first\_name, last\_name, email, password, **data**)  **VALUES** (**'Max'**, **'Mustermann'**, **'m.mustermann@mail.com'**, **'pw1'**,   **'<widget>email:m.mustermann@mail.com:pw1</widget>  <widget>calendar:m.mustermann@mail.com:pw1</widget>  <widget>weather</widget>  <widget>clock</widget>'**) |

Codeabschnitt 39: SQL-Statement zum Einfügen eines Benutzers (Database.php)

Nachdem der Query generiert wurde, wird dieser mit *prepare(...)* vorbereitet und mit *execute()* ausgeführt. Danach wird die Ausführung gestoppt und ein möglicher Fehler als Return-Wert zurückgegeben.

* 1. **Schnittstellendateien**

Das Frontend des PHP-Skripts bilden verschiedenste Schnittstellen-Dateien. Diese werden mithilfe des HTTP-Protokolls von der Smartphone-App aufgerufen. Für jede Operation, die in der App getätigt wird, gibt es eine eigene Schnittstellen-Datei. Diese Dateien sind *signin.php*, *signup.php*, *setdata.php*, *getdata.php*. Jede dieser einzelnen Dateien besitzt eine eigene Konfigurationsdatei, welche mit dem Befehl *require* inkludiert wird. Wichtig bei den vorher genannten Dateien ist, dass sie nicht mit den *interface\_*-Dateien verwechselt werden. Das Wort „Interface“ bezieht sich hierbei auf eine grafische Benutzerschnittstelle und nicht auf eine Frontend-Schnittstelle.

* + 1. Reflecton/api/signup.php

|  |
| --- |
| **<?php****require '../includes/Database.php'**; **require '../includes/config.php'**; **require '../includes/signup\_config.php'**;  **global** $columnElementsSignUp; **global** $rowElements;  **if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**) {  $db = **new** Database(***DB\_HOST***, ***DB\_USER\_NAME***, ***DB\_USER\_PWD***, ***DB\_NAME***, ***TABLE\_NAME***,   $columnElementsSignUp);  $db->connectToDatabase();  **for** ($i = 0; $i < *count*($columnElementsSignUp); $i++) {  $rowElements[$i] = $\_POST[$columnElementsSignUp[$i]->getName()];  }  $flag1 = $db->createTable();  **echo** $flag1;  $flag2 = $db->insertRow($rowElements);  **echo** $flag2;  $db->closeConnectionToDatabase(); } |

Codeabschnitt 40: Schnittstelle zum Erstellen eines Kontos (signup.php)

Die Grundstruktur ist in jeder der einzelnen Schnittstellen-Dateien dieselbe. Nach diversen Importvorgängen mit *require* und Definition globaler Variablen, stellt sich die Frage, ob die Anfragemethode *REQUEST\_METHOD* der *POST*-Methode entspricht. Dies muss selbstverständlich in unserer Smartphone-App eingestellt werden. Im Block dieser if-Anweisung wird zuerst eine Instanz der Klasse *Database* erstellt und eine Verbindung zum Datenbankserver aufgebaut. Um nun die vom Smartphone gesendeten Werte zu verarbeiten ist die for-Schleife notwendig. Hierbei ist dringend darauf zu achten, dass das Smartphone die Parameter mit den gleichen Benennungen schickt, wie sie auch in *$columnElementsSignUp[$i]->getName()* zu finden sind. Ansonsten funktioniert *$\_POST[...]* nicht. Die vom Smartphone gesendeten Werte werden ordnungsgemäß im Array *$rowElements* gespeichert. Dies ist eine Art Abgleich eines bestimmten Datensatzes. Die Struktur der Tabelle ist in *$columnElementsSignIn* gespeichert, sowie die benötigten Werte in *$rowElements*. Mit diesen Werten kann eine Tabelle mit *createTable()* erstellt werden, sowie ein Datensatz mit *insertRow(...)* angelegt werden. Bei Aufruf dieses Skripts wird ein neuer Benutzer angelegt. Die Verbindung zur Datenbank wird ordnungsgemäß mithilfe der Funktion *closeConnectionToDatabase()* beendet.

* + 1. Reflecton/api/signin.php

|  |
| --- |
| **<?php****require '../includes/Database.php'**; **require '../includes/config.php'**; **require '../includes/signin\_config.php'**; **global** $columnElementsSignIn; **global** $rowElements;  **if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**) {  $db = **new** Database(***DB\_HOST***, ***DB\_USER\_NAME***, ***DB\_USER\_PWD***, ***DB\_NAME***, ***TABLE\_NAME***,   $columnElementsSignIn);  $db->connectToDatabase();  **for** ($i = 0; $i < *count*($columnElementsSignIn); $i++) {  $rowElements[$i] = $\_POST[$columnElementsSignIn[$i]->getName()];  }  $flag1 = $db->selectRow($rowElements);  **echo** $flag1;  $db->closeConnectionToDatabase(); } |

Codeabschnitt 41: Schnittstelle zum Anmelden (signin.php)

Der einzige Unterschied dieses Codes im Gegensatz zu jenem in *signup.php* ist, dass hier andere Methoden verwendet werden. Mit der Funktion *selectRow(...)* wird ein einziger Datensatz ausgewählt. Wenn also ein Datensatz mithilfe von E-Mail und Passwort gefunden wurde, steht in *$flag1* ein Einser und der Anmeldevorgang ist somit geglückt.

* + 1. Reflecton/api/setdata.php

|  |
| --- |
| **<?php****require '../includes/Database.php'**; **require '../includes/config.php'**; **require '../includes/setdata\_config.php'**;  **global** $columnElementsSetData; **global** $rowElements; **if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**) {  $db = **new** Database(***DB\_HOST***, ***DB\_USER\_NAME***, ***DB\_USER\_PWD***, ***DB\_NAME***, ***TABLE\_NAME***,   $columnElementsSetData);  $db->connectToDatabase();  **for** ($i = 0; $i < *count*($columnElementsSetData); $i++) {  $rowElements[$i] = $\_POST[$columnElementsSetData[$i]->getName()];  }$flag1 = $db->updateRowSearchByFirstElement($rowElements);  **echo** $flag1;  $db->closeConnectionToDatabase(); } |

Codeabschnitt 42: Schnittstelle zum Aktualisieren eines Datensatzes (setdata.php)

Bei *setdata.php* wird, wie der Name schon verrät, ein Datum gesetzt. Dies ist in unserem Fall das Feld *data*. Die wichtigste Funktion die in diesem Block aufgerufen ist *updateRowSearchByFirstElement(...)*. Dabei wird das Feld *data*, welches mithilfe des Schlüssels *email* aufgesucht wird, mit der aktuellen Widget-Anordnung aktualisiert.

* + 1. Reflecton/api/getdata.php

|  |
| --- |
| **<?php****require '../includes/Database.php'**; **require '../includes/config.php'**; **require '../includes/getdata\_config.php'**;  **global** $columnElementsGetData; **global** $rowElements; **global** $resultField;  **if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**) {  $db = **new** Database(***DB\_HOST***, ***DB\_USER\_NAME***, ***DB\_USER\_PWD***, ***DB\_NAME***, ***TABLE\_NAME***,   $columnElementsGetData);  $db->connectToDatabase();   **for** ($i = 0; $i < *count*($columnElementsGetData); $i++) {  $rowElements[$i] = $\_POST[$columnElementsGetData[$i]->getName()];  }  $resultArray = $db->selectRowReturnResult($rowElements);  **if** (*count*($resultArray) != 0) {  **echo** $resultArray[$resultField];  }  **else** {  **echo** 0;  }  $db->closeConnectionToDatabase(); } |

Codeabschnitt 43: Schnittstelle zum Auswählen eines Datensatzes (getdata.php)

Die Schnittstelle *getdata.php* wird für das Auswählen eines bestimmten Datensatzes verwendet. Dies wird vor allem bei der Erstanordnung beim Starten der App benötigt. Die wichtigste Funktion in diesem Block ist *selectRowReturnResult(...)*. Jene ist zum Auswählen eines Datensatzes verantwortlich.

* 1. **HTML-Interface**

Die *interface\_-*Dateien haben für die Kommunikation zwischen Smartphone und Datenbank eigentlich keinen Zweck. Vielmehr sind sie eine grafische Schnittstelle am Webbrowser zur Datenbank. Da das PHP-Skript vor der iOS-App entwickelt wurde, war noch keine geeignete Testumgebung existent. Deshalb wurden die Interface-Dateien hinzugefügt, um vorläufige Tests vorzunehmen. Da alle vier Interface-Dateien im api-Ordner fast gleich sind, wird zur genaueren Beschreibung nur die Datei *interface\_signin.php* verwendet. Der folgende Programmcode zeigt die Umsetzung dieser Testumgebung.

* + 1. Reflecton/includes/interface\_signin.php

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta name="viewport" content="width=device-width, initial-scale=1"**>  <**link rel="stylesheet" href="http://www.w3schools.com/lib/w3.css"**>  <**title**>Sign In</**title**> </**head**> <**body class="w3-container" background="background1.jpg"**> <**header class="w3-container w3-blue w3-round-large w3-card-4"**>  <**h1**>Sign In</**h1**> </**header**> <**form method="post" action="signin.php" class="w3-form w3-card-4"**>  <**div class="w3-input-group"**>  <**p align=center**>  **<?php  require '../includes/signin\_config.php'**;  **global** $columnElementsSignIn;  **for** ($i = 0; $i < *count*($columnElementsSignIn); $i++) {  **echo** $columnElementsSignIn[$i]->getShownName() . **':<br>'**;  **echo '<input type="text" name="'**. $columnElementsSignIn[$i]->getName()  .**'"><br><br>'**;  }  **echo '<input type="submit" class="w3-btn w3-blue w3-round-large"  value="Submit">'**;  **?>** </**p**>  </**div**> </**form**> <**footer class="w3-container w3-blue w3-round-large w3-card-4"**>  <**p**>Pinnovations (2017)</**p**> </**footer**> </**body**> </**html**> |

Codeabschnitt 44: HTML-Interface zum Anmelden (interface\_signin.php)

Zwischen relativ einfacher HTML-Codierung findet sich ein PHP-Block, der von besonderer Bedeutung ist. Das globale Array *$columnElementsSignIn* vom Typ *Column*-Array, welches in der *signin\_config.php* deklariert ist, wird in der for-Schleife durchlaufen. Dabei wird das anzuzeigende Element durch die Funktion *getShownName()* bereitgestellt. Danach wird nach einem Zeilenumbruch das zugehörige Textfeld hinzugefügt. Dieses erhält mittels der Funktion *getName()* den exakt gleichen Namen wie die Spalte in der Datenbank.

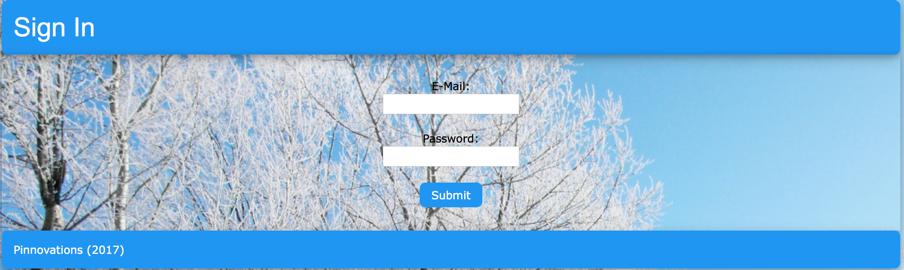


Abbildung 30: HTML-Interface

Der obige Screenshot zeigt das Log In-HTML-Interface, welches zum Austesten des PHP-Skriptes nötig war. Dieses verwendet ein Template der Seite *w3schools.com*.[[1]](#footnote-2)

1. **PHP-Interface zum Pi**

Mithilfe des PHP-Interfaces, welches vom Raspberry Pi gehostet wird, kann der Spiegel per Smartphone-App mit dem Heimnetzwerk verbunden werden. Dabei wird der Raspberry Pi zwischen zwei Modi geschaltet werden. Entweder als Router oder als Client im Heimnetzwerk.

Das PHP Interface wird von der Smartphone-App aufgerufen, um den Raspberry Pi mit dem Internet verbinden zu können. Somit muss am Spiegel kein zusätzliches Gerät angeschlossen werden um beispielsweise das Passwort für das jeweilige WLAN-Netzwerk eingeben zu können.

* 1. **Vorbereitungen**
     1. Vorbereitungen am Raspberry Pi

Um am Raspberry PI ein PHP-Skript ausführen zu können, muss PHP7 zuerst auf dem Raspberry PI installiert werden. Dies wird über die sogenannte *apt-url*, welche für Installationen unter Linux sehr häufig verwendet wird.

Um zuerst die Pakete auf den aktuellen Stand zu bringen, werden diese zuerst aktualisiert, was mit dem folgenden Befehl ausgeführt werden kann.

|  |
| --- |
| sudo apt-get update |

Codeabschnitt 45: Update des Raspberry Pi

Sobald die Pakete aktualisiert sind, müssen diese noch installiert bzw. auf den neusten Stand gebracht werden. Hierfür wird der *upgrade* (Aktualisieren)-Befehl bei der *apt-url* verwendet.

|  |
| --- |
| sudo apt-get upgrade |

Codeabschnitt 46: Installation der Aktualisierungen am Raspberry Pi

Zuerst werden die neuesten Pakete von PHP installiert, wofür ein Interpreter benötigt wird. Mit PHP wird ein Skript erstellt, welches anschließend die eingegebenen Parameter an den Raspberry Pi sendet. Der folgende Befehl zeigt wie die Installation ausgeführt wird:

|  |
| --- |
| sudo apt-get install php7.0 |

Codeabschnitt 47: Installation von PHP am Raspberry Pi

Damit das PHP-Skript später auch über das Internet erreichbar ist, wird ein *Apache2*-Webserver installiert. Hierfür müssen zuerst die Pakete installiert werden, was mit dem nachfolgenden Befehl durchgeführt wird:

|  |
| --- |
| sudo apt-get install apache2 |

Codeabschnitt 48: Installation Apache-Server am Raspberry Pi

Sobald die Installation abgeschlossen ist, wird eine Testdatei erstellt, um die Funktion des Servers zu überprüfen. Dafür wird im Hauptordner des Webservers ein PHP-Skript erstellt, welches mit folgendem Befehl durchgeführt wird:

|  |
| --- |
| sudo nano /var/www/html/info.php |

Codeabschnitt 49: Erstellung einer Testdatei zum Testen des Servers

Anschließend öffnet sich in der Kommandozeile ein Editor, mit dem anschließend ein Inhalt in die Datei eingefügt werden kann. In dieser Skript-Datei, welche nur zu Funktionszwecken verwendet wird, wird der Text *server is online* ausgegeben, was mit folgendem PHP-Befehl erzielt wird:

|  |
| --- |
| <?php echo “server is online</br>”;phpinfo; ?> |

Codeabschnitt 50: Code zum Testen des Servers

Wenn nun die IP-Adresse des Raspberry PI in der Adressleiste eines im selben Netzwerk vorhandenen Computers und anschließend */info.php* hinzugefügt wird, sollte der im vorherigen Schritt definierte Text angezeigt werden.

* + 1. Vorbereitungen für Hotspot

Im Auslieferungszustand des Spiegels ist kein Netzwerk in der wpa\_supplicant.conf-Datei, welche die Verwaltung der WLAN-Netzwerke steuert, eingetragen. Damit ein Benutzer trotzdem mit dem Spiegel eine Verbindung aufbauen und mit dem Heimnetzwerk verbinden kann, muss dieser zuvor einen Hotspot aufbauen.

Um den Spiegel mit dem Heimnetzwerk verbinden zu können, muss sich Benutzer davor mit der Smartphone-App eine Verbindung zu dem ausgestrahlten Netzwerk des Raspberry Pi aufbauen. Hierfür werden weitere Pakete benötigt, welche im Laufe des Kapitels installiert werden.

Zuerst wird die *hostapd*-Anwendung benötigt. Wird ein bekanntes Netzwerk in der Umgebung des Controllers gefunden, so verbindet sich dieser direkt zu diesem. Ist dies nicht der Fall, so strahlt er sein eigenes Netzwerk aus. Dies wird mit der nachfolgenden Anwendung realisiert, die mit folgendem Befehl installiert wird.

|  |
| --- |
| sudo apt-get -y install hostapd |

Codeabschnitt 51: Installation von hostapd am Raspberry Pi

Als nächstes wird das Softwarepaket *dnsmasq* installiert. Mithilfe dieses Paketes kann der Raspberry Pi als kleiner DNS (Domain Name Service) bzw. DHCP-Server verwendet werden. Installiert wird dieser mit folgendem Befehl:

|  |
| --- |
| sudo apt-get -y install dnsmasq |

Codeabschnitt 52: Installation der dnsmasq-Pakete am Raspberry Pi

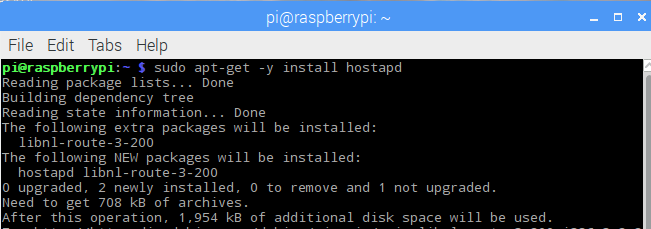


Abbildung 31: Installationsprozess des hostapd-Paket

Nach der Installation werden die beiden Services deaktiviert, damit sie entsprechend konfiguriert werden können. Dies wird mit den beiden nachfolgenden Befehlen durchgeführt:

|  |
| --- |
| sudo systemctl disable hostapd |

Codeabschnitt 53: Deaktivierung des hostapd-Service

|  |
| --- |
| sudo systemctl disable dnsmasq |

Codeabschnitt 54: Deaktivierung des dnsmasq-Service

Nun kann die Hostspot-Konfiguration in den betreffenden Dateien vorgenommen werden. Diese Dateien beinhalten die Zugangsdaten (SSID und Passwort).

Nun werden die *hostapd*-Pakete mithilfe des folgenden Befehls editiert:

|  |
| --- |
| sudo nano /etc/hostapd/hostapd.conf |

Codeabschnitt 55: Öffnen der Konfigurationsdatei von hostapd

Alle in dieser Datei durchgeführten Installationen werden an den auszustrahlenden WLAN-Hotspot angewendet. In der Datei sollten dieselben Einstellungen angeführt werden, wie der nachfolgende Codeabschnitt anzeigt. Netzwerkname und Passwort können individuell gewählt werden.

|  |
| --- |
| interface=wlan0  driver=nl88211  ssid=Reflecton  hw\_mode=g  channel=6  wmm\_enabled=0  macaddr\_acl=0  auth\_algs=1  ignore\_broadcast\_ssid=0  wpa=2  wpa\_passphrase=reflecton  wpa\_key\_mgmt=WPA-PSK  wpa\_pairwise=TKIP  rsn\_pairwise=CCMP |

Codeabschnitt 56: Inhalt der Konfigurationsdatei

Nun sind die grundlegenden Einstellungen bezüglich Netzwerkname, Passwort und Sicherheit vorgenommen, welche dafür sorgen, dass der Hotspot gefunden wird und sich ein Benutzer damit verbinden kann. Nun muss noch in der Hauptkonfigurationsdatei der Dateipfad zur eben editierten Konfigurationsdatei übergeben werden. Somit werden die Einstellungen auch wirksam, wenn der Hotspot gestartet wird. Dies wird über die folgenden beiden Variablen DAEMON\_CONF, welcher der Pfad der eben editierten Konfigurationsdatei übergeben und DAEMON\_OPT, vor der das Kommentarzeichen # entfernt wird, ausgeführt. Der Wert der DAEMON\_OPT -Variable wird nicht editiert.

|  |
| --- |
| DAEMON\_CONF=“/etc/hostapd/hostapd.conf“  DAEMON\_OPTS=“….“ |

Codeabschnitt 57: Übergabe des Pfades der Konfigurationsdatei

Sobald die Einstellungen gespeichert sind, ist die Konfiguration, welche für den ausgestrahlten Hotspot benötigt wird, abgeschlossen.

In den nächsten Schritten wird der DNS bzw. DHCP-Server auf dem Raspberry Pi konfiguriert. Dieser Server ermöglicht die Zuweisung einer Netzwerkkonfiguration des Clients durch den Server. Diese Einstellungen werden ebenfalls mithilfe einer Datei durchgeführt, wobei hier bereits eine Standardkonfiguration vorhanden ist, welche mit folgendem Befehl angezeigt werden kann:

|  |
| --- |
| sudo nano /etc/dnsmasq.conf |

Codeabschnitt 58: Öffnen der Konfigurationsdatei von dnsmasq

In dieser Datei wird der unterste Bereich bearbeitet. Der anschließende Ausschnitt zeigt, welche Einstellungen in der Konfigurationsdatei editiert werden. Im Falle des ausgestrahlten Netzwerkes, wird ein Klasse-C Netz gewählt, da eine maximale Host-Anzahl von 254 ausreichen sollte.

|  |
| --- |
| #AutoHostpot config  interface=wlan0  no-resolv  bind-dynamic  server=8.8.9.8  domain-needed  bogus-priv  dhcp-range=192.168.50.150,192.168.50.200,255.255.255.0,12h |

Codeabschnitt 59: Konfiguration des dnsmasq

Da nun auch der DNS-Server fertig konfiguriert wurde, wird noch IP-Forwarding eingestellt. Dabei wird das Paket an den Router gesendet, anschließend analysiert, wohin das Paket geleitet werden soll und zum nächsten Router gesendet. Dafür wird mit folgendem die Konfigurationsdatei der Schnittstellen angezeigt:

|  |
| --- |
| sudo nano /etc/network/interfaces |

Codeabschnitt 60: Anzeigen der Netzwerkschnittstellen

Nun werden die ersten fünf Zeilen der Datei kopiert, welche ähnlich wie die Folgenden aussehen sollten:

|  |
| --- |
| # interfaces(5) file used by ifup(8) and ifdown(8)  # Please note that this file is written to be used with dhcpcd  # For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'  # Include files from /etc/network/interfaces.d:  source-directory /etc/network/interfaces.d |

Codeabschnitt 61: Erste fünf Zeilen der Interfaces-Datei

Für den Fall, dass die Konfiguration nicht funktioniert wird eine Sicherungsdatei mit folgendem Befehl angefertigt:

|  |
| --- |
| sudo cp /etc/network/interfaces /etc/network/interfaces-backup |

Codeabschnitt 62: Erstellung einer Sicherungsdatei der Netzwerkschnittstellen

Im Hotspot-Mode des Raspberry Pi muss *IP-Forwarding* aktiviert sein, um eine Verbindung zum Internet aufbauen zu können, wenn beispielsweise ein LAN-Kabel angesteckt ist. Der Modus, in dem sich der Raspberry Pi aktuell befindet ist abhängig davon, ob der Raspberry Pi ein vertrautes WLAN-Netzwerk finden kann. Bevor ein Skript zum automatischen Umschalten der Modi allerdings geschrieben werden kann, muss noch das *IP-Forwarding* standardmäßig aktiviert werden. Mit folgendem Befehl wird der Inhalt der dafür benötigten Konfigurationsdatei angezeigt:

|  |
| --- |
| sudo nano /etc/sysctl.conf |

Codeabschnitt 63: Öffnen der Konfigurationsdatei für IP-Forwarding

Nun wird nach der folgenden Zeile in der Datei gesucht, um dabei das Kommentarzeichen zu entfernen. Diese Zeile sollte nun wie folgt aussehen:

|  |
| --- |
| #Uncomment the next line to enable packet forwarding for IPv4  net.ipv4.ip\_forward=1 |

Codeabschnitt 64: Aktivierung von IP-Forwarding

Im nächsten Schritt wird eine Service-Datei erstellt, die das Autohotspot-Skript automatisch beim Start des Raspberry Pi startet. Die dafür benötigte Datei wird mit folgendem Befehl erzeugt:

|  |
| --- |
| sudo nano /etc/systemd/system/autohotspot.service |

Codeabschnitt 65: Erstellung eines Autohotspot-Skripts

Nun wird die Datei mit dem folgenden Code gefüllt. Dieses Skript startet automatisch einen Hotspot, wenn kein bekanntes WLAN-Netzwerk in Reichweite ist.

|  |
| --- |
| [Unit]  Description=Automatically generates an internet Hotspot when a valid  ssid is not in range  After=multi-user.target  [Service]  Type=oneshot  RemainAfterExit=yes  ExecStart=/usr/bin/autohotspotN  [Install]  WantedBy=multi-user.target |

Codeabschnitt 66: Inhalt Autohotspot-Skript

Der nun eben erstellte Service kann mit dem nachfolgenden Befehl gestartet werden:

|  |
| --- |
| sudo systemctl enable autohotspot service |

Codeabschnitt 67: Starten des Autohotspot-Skriptes

* + 1. Auto-Hotspot Skript

Dieses Skript ist ein elementarer Baustein des Spiegels, da es ohne diesem nicht möglich wäre, diesen mit dem Internet zu verbinden, um die benötigten Daten herunterzuladen. Dieses Skript überprüft grundsätzlich die bereits bekannten Netzwerke, welche in der wpa\_supplicant.conf-Datei am Raspberry Pi abgespeichert sind, mit denen, die aktuell in Reichweite sind. Ist ein bekanntes Netzwerk in der Umgebung, so verbindet sich der Raspberry Pi mit dem Hotspot als Client. Ist kein bekanntes Netzwerk in der Nähe, so strahlt der Raspberry Pi sein eigenes Netzwerk aus, wie der folgende Code zeigt. Der nachfolgende Code stammt von der Website *Raspberry Connect*[[2]](#footnote-3):

|  |
| --- |
| wifidev="wlan0" #device name to use. Default is wlan0.  #use the command: iw dev ,to see wifi interface name  IFSdef=$IFS  cnt=0  wpassid=$(awk '/ssid="/{ print $0 }' /etc/wpa\_supplicant/wpa\_supplicant.conf | awk -F'ssid=' '{ print $2 }' ORS=',' | sed 's/\"/''/g' | sed 's/,$//')  IFS=","  ssids=($wpassid)  IFS=$IFSdef #reset back to defaults  mac=()  ssidsmac=("${ssids[@]}" "${mac[@]}") #combines ssid and MAC for checking  createAdHocNetwork() {  ip link set dev "$wifidev" down  ip a add 192.168.50.5/24 brd + dev "$wifidev"  ip link set dev "$wifidev" up  iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE  iptables -A FORWARD -i eth0 -o "$wifidev" -m state --state RELATED,ESTABLISHED -j ACCEPT  iptables -A FORWARD -i "$wifidev" -o eth0 -j ACCEPT  systemctl start dnsmasq  systemctl start hostapd  echo 1 > /proc/sys/net/ipv4/ip\_forward  }  KillHotspot()  {  echo "Shutting Down Hotspot"  ip link set dev "$wifidev" down  systemctl stop hostapd  systemctl stop dnsmasq  iptables -D FORWARD -i eth0 -o "$wifidev" -m state --state RELATED,ESTABLISHED -j ACCEPT  iptables -D FORWARD -i "$wifidev" -o eth0 -j ACCEPT  echo 0 > /proc/sys/net/ipv4/ip\_forward  ip addr flush dev "$wifidev"  ip link set dev "$wifidev" up  }  ChkWifiUp()  {  echo "Checking WiFi connection ok"  sleep 10 #give time for connection to be completed to router  if ! wpa\_cli -i "$wifidev" status | grep 'ip\_address' >/dev/null 2>&1  then #Failed to connect to wifi (check your wifi settings, password etc)  echo 'Wifi failed to connect, falling back to Hotspot.'  wpa\_cli terminate "$wifidev" >/dev/null 2>&1  createAdHocNetwork  fi  }  FindSSID()  {  #Check to see what SSID's and MAC addresses are in range  ssidChk=('NoSSid')  i=0; j=0  until [ $i -eq 1 ] #wait for wifi if busy, usb wifi is slower.  do  ssidreply=$((iw dev "$wifidev" scan ap-force | egrep "^BSS|SSID:") 2>&1) >/dev/null 2>&1  if echo "$ssidreply" | grep "No such device (-19)" >/dev/null 2>&1; then  NoDevice  elif ! echo "$ssidreply" | grep "resource busy (-16)" >/dev/null 2>&1 ;then  i=1  elif (($j >= 5)); then #if busy 5 times goto hotspot  ssidreply=""  i=1  else #see if device not busy in 2 seconds  j=$((j = 1))  sleep 2  fi  done  for ssid in "${ssidsmac[@]}"  do  if (echo "$ssidreply" | grep "$ssid") >/dev/null 2>&1  then  #Valid SSid found, passing to script  ssidChk=$ssid  return 0  else  #No Network found, NoSSid issued"  ssidChk='NoSSid'  fi  done  }  NoDevice()  {  #if no wifi device,ie usb wifi removed, activate wifi so when it is  #reconnected wifi to a router will be available  echo "No wifi device connected"  wpa\_supplicant -B -i "$wifidev" -c /etc/wpa\_supplicant/wpa\_supplicant.conf >/dev/null 2>&1  exit 1  }  FindSSID  #Create Hotspot or connect to valid wifi networks  if [ "$ssidChk" != "NoSSid" ]  then  echo 0 > /proc/sys/net/ipv4/ip\_forward #deactivate ip forwarding  if systemctl status hostapd | grep "(running)" >/dev/null 2>&1  then #hotspot running and ssid in range  KillHotspot  echo "Hotspot Deactivated, Bringing Wifi Up"  wpa\_supplicant -B -i "$wifidev" -c /etc/wpa\_supplicant/wpa\_supplicant.conf >/dev/null 2>&1  ChkWifiUp  elif { wpa\_cli -i "$wifidev" status | grep 'ip\_address'; } >/dev/null 2>&1  then #Already connected  echo "Wifi already connected to a network"  else #ssid exists and no hotspot running connect to wifi network  echo "Connecting to the WiFi Network"  wpa\_supplicant -B -i "$wifidev" -c /etc/wpa\_supplicant/wpa\_supplicant.conf >/dev/null 2>&1  ChkWifiUp  fi  else #ssid or MAC address not in range  if systemctl status hostapd | grep "(running)" >/dev/null 2>&1  then  echo "Hostspot already active"  elif { wpa\_cli status | grep "$wifidev"; } >/dev/null 2>&1  then  echo "Cleaning wifi files and Activating Hotspot"  wpa\_cli terminate >/dev/null 2>&1  ip addr flush "$wifidev"  ip link set dev "$wifidev" down  rm -r /var/run/wpa\_supplicant >/dev/null 2>&1  ip link set dev "$wifidev" up  createAdHocNetwork  else #"No SSID, activating Hotspot"  createAdHocNetwork  fi  fi |

Codeabschnitt 68: Autohotspot.service

* + 1. Vorbereitungen für PHP-Interface

Damit das PHP-Interface die Parameter für das WLAN in die Konfigurationsdatei schreiben kann, müssen die Zugriffsrechte bearbeitet werden. Mithilfe des nachfolgenden Befehls wird auf den Benutzer *www-data* verwiesen, da dies der Benutzer ist, mit welchem der Apache-Webserver auf die Dateien im System schreiben kann. Diesem werden hierbei die Zugriffsrechte auf die *wpa-supplicant.conf* erteilt, welche sich im Verzeichnis */etc/wpa\_supplicant* befinden.

|  |
| --- |
| sudo chown www-data:www-data wpa\_supplicant.conf |

Codeabschnitt 69: Erteilung der Benutzerrechte www-data

Wenn der WLAN-Hotspot ausgestrahlt werden soll, sieht das Konfigurationsfile entsprechend des nachfolgenden Screenshots aus. Hierbei sind der Name des auszustrahlenden Netzwerkes, das verwendete Passwort und die Verschlüsselungsmethode sichtbar. Mithilfe des Zusatzes *off* wird definiert, dass es sich hierbei um ein Netzwerk handelt, welches vom Raspberry Pi ausgestrahlt wird.

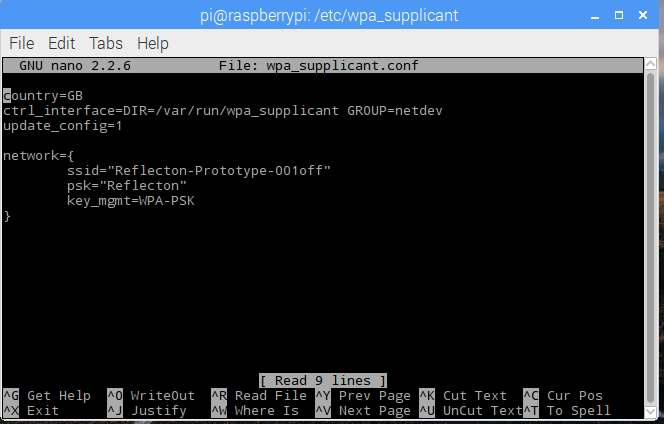


Abbildung 32: Konfigurationsdatei der Netzwerke (*wpa\_supplicant.conf*)

* 1. **WLAN-Konfiguration mithilfe von PHP**

Um nun dem Spiegel die Internetverbindung zu ermöglichen, wird eine PHP-Datei erzeugt, mit welcher der Inhalt der Konfigurationsdatei mit den Eingabewerten des Benutzers befüllt wird. Um mit einem Netzwerk eine Verbindung herstellen zu können, ist am Raspberry Pi eine *wpa\_supplicant.conf*-Datei verfügbar, in der von allen bekannten WLAN-Netzwerken Netzwerkname, Passwort und Verschlüsselungstyp eingetragen wird. Dieses PHP-Skript ist auf dem Webserver vom Raspberry Pi hochgeladen, da somit auch die Datenübertragung zwischen Smartphone-App und Raspberry Pi möglich ist. Wenn der Benutzer in der App in den Sektor WLAN-Konfiguration wechselt, wird ein Formular anzeigt, in dem der Netzwerkname, Passwort und Verschlüsselungstyp eingegeben werden können, wie in folgendem Code erkennbar ist:

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**>  <**head**>  <**meta name="viewport" content="width=device-width, initial-scale=1"**>  <**link rel="stylesheet" href="http://www.w3schools.com/lib/w3.css"**>  <**title**>Setup</**title**> </**head**>  <**header class="w3-container w3-blue w3-round-large w3-card-4"**>  <**h1**>Setup</**h1**> </**header**>  <**form method="post" action="setup.php" class="w3-form w3-card-4"**>  <**div class="w3-input-group"**>  <**p align=center**>  **<?php  echo 'E-Mail: '** . **':<br>'**;  **echo '<input type="text" name="email"><br><br>'**;  **echo 'SSID: '** . **':<br>'**;  **echo '<input type="text" name="ssid"><br><br>'**;  **echo 'PSK: '** . **':<br>'**;  **echo '<input type="text" name="psk"><br><br>'**;  **echo '<input type="submit" class="w3-btn w3-blue w3-round-large" value="Submit">'**;  **?>** </**p**>  </**div**> </**form**>  <**footer class="w3-container w3-blue w3-round-large w3-card-4"**>  <**p**>Pinnovations (2017)</**p**> </**footer**>  </**body**>  </**html**> |

Codeabschnitt 70: Wifi-Config-Interface in HTML

Bevor die Benutzereingaben übergeben werden, wird zuerst überprüft, ob überhaupt ein Datentransfer stattgefunden hat. Anschließend werden die Benutzereingaben übergeben und in die *wpa\_supplicant.conf* bei den entsprechenden Parametern eingefügt. Danach werden noch die Berechtigungen so angepasst, damit die Datei anschließend auch gespeichert werden kann. Außerdem wird auch noch die E-Mail-Adresse des Benutzers in einem eigenen Textdokument abgespeichert, damit die Anordnung auf alle Fälle vom richtigen Benutzer aus der Datenbank ausgelesen wird. Nachdem der Prozess erfolgt ist, wird eine Antwort (in diesem Fall *1*) zurückgesetzt, wodurch der Benutzer eine Rückmeldung bekommt, dass die Eintragung der Parameter erfolgreich war. Danach wird der Raspberry Pi mit dem Kommando *sudo reboot* neu gestartet und die Änderungen werden somit wirksam. Danach beginnt sich der im vorherigen Kapitel beschriebene Service zu aktivieren und der Prozess beginnt von Neuem.

|  |
| --- |
| **<?php** *// check if http method is post* **if**($\_SERVER[**'REQUEST\_METHOD'**]==**'POST'**) {  *// get post parameters* $email = $\_POST[**'email'**];  $ssid = $\_POST[**'ssid'**];  $psk = $\_POST[**'psk'**];   *// path of wpa\_supplicant* $pathWiFi = **'/etc/wpa\_supplicant/wpa\_supplicant.conf'**;   *// get content of wpa\_supplicant* $contentWiFi = *file\_get\_contents*($pathWiFi);   *// change content of wpa\_supplicant* $ssid\_start = *strpos*($contentWiFi, **'ssid="'**) + 6;  $ssid\_end = *strpos*($contentWiFi, **'"'**, $ssid\_start);  $psk\_start = *strpos*($contentWiFi, **'psk="'**) + 5;  $psk\_end = *strpos*($contentWiFi, **'"'**, $psk\_start);  $new\_contentWiFi = *substr*($contentWiFi, 0, $ssid\_start)  . $ssid  . *substr*($contentWiFi, $ssid\_end, $psk\_start - $ssid\_end)  . $psk  . *substr*($contentWiFi, $psk\_end);   *// write in file (permission needed: sudo chown www-data:www-data wpa\_supplicant.conf)  file\_put\_contents*($pathWiFi, $new\_contentWiFi);   *// path of email* $pathEmail = **'/home/pi/Documents/Reflecton/Email.txt'**;   *// write in file (permission needed: sudo chown www-data:www-data Email.txt)  file\_put\_contents*($pathEmail, $email);   *// send ok flag* **echo** 1;   *// reboot mirror  exec*(**'sudo reboot'**);  } **?>** |

Codeabschnitt 71: Wifi-Config.php

Mithilfe eines Smartphones kann der Spiegel über das PHP Interface mit einem WLAN-Netzwerk konfiguriert werden. Wird das PHP-Interface zum Konfigurieren verwendet, so werden die Parameter für Netzwerkname (SSID) und Passwort (PSK) mit den Benutzereingaben durch die bereits vorhandenen ersetzt. Die Änderungen werden nach einem Neustart des Controllers wirksam, welcher automatisch nach erfolgreichem Übergeben der Parameter durch das PHP-Interface ausgeführt wird.

1. **User Interface**

Das Herzstück von Reflecton stellt die grafische Benutzeroberfläche dar, welche mit der Programmiersprache Java entwickelt wurde.

* 1. **Struktur**

Im unten dargestellten Bild ist die Struktur des Java-UI-Programms ersichtlich.

|  |  |
| --- | --- |
| Abbildung 33: Struktur | Der erste Ordner *assets* enthält alle notwendigen Bilder, sowie Templates. Beispiele dafür sind geeignete Bilder zum Anzeigen des aktuellen Wetters, oder auch Bilder für den Hintergrund der spezifischen Widgets.  Im Hauptordner *com.pinnovations* befinden sich weitere Ordner, in denen sich verschiedenste Java-Dateien befinden. Neben etlichen Konfigurationsdateien finden sich eine Datenbankschnittstelle, eine Downloadschnittstelle, verschiedenste UI-Klassen, sowie etliche Hilfsklassen wieder.  Die *main*-Funktion befindet sich in der Datei *ReflectonUI.java*. Sie ist Dreh- und Angelpunkt des ganzen Programmes.  Für das Programm ebenfalls wichtig sind die Ordner *files*, *fonts* und *scripts*. Sie stellen allesamt eine Erweiterung zum Java-Programm dar. Die hier sichtbare *GeoLiteCity.dat* ist für die Bibliothek des Wetterdienstes notwendig.[[3]](#footnote-4) Hingegen ist das Skript *IcloudCalendar.py* für das Auslesen der Daten eines iCloud-Kalenders verantwortlich. |

* 1. **Vorbereitungen**

Da das Java-Programm auf dem Raspberry Pi laufen soll, sind einige Schritte notwendig. Es werden verschiedenste Installationsvorgänge und Änderungen von bestimmten Dateien fällig.

* + 1. Raspberry Pi Konfiguration

Der Raspberry Pi muss zuerst an einen Monitor mit HDMI, sowie an eine Stromversorgung angeschlossen werden. Um ihn bedienen zu können, muss über USB eine Tastatur, sowie eine Maus angeschlossen werden.

Mit einem Klick auf das Beerensymbol auf der linken oberen Seite kann nun *Raspberry Pi Configuration* unter dem Reiter *Preferences* ausgewählt werden. Danach wechselt man in die Registerkarte *Interfaces* und aktiviert SSH.

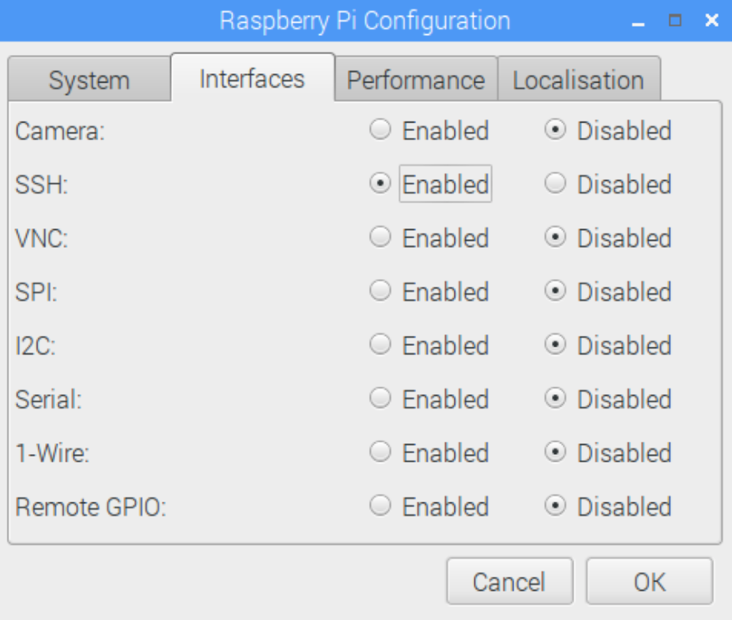


Abbildung 34: Konfigurationsmenü

Außerdem sollte bereits Serial aktiviert werden, da dies später zur Kommunikation zwischen Arduino und Pi benötigt wird.

Der Raspberry Pi ist in ein beliebiges Netzwerk via WiFi zu verbinden. Der Computer, auf jenem das Java-Programm entwickelt wird, muss ebenfalls in dieses Netzwerk verbunden werden. Mit einem Programm wie z.B. „LanScan“ für MacOS kann nun die IP-Adresse des Pis herausgefunden werden. Über folgenden Befehl kann nun über das Terminal in Linux oder MacOS mit dem Pi interagiert werden:

|  |
| --- |
| ssh pi@your-ip-address |

Codeabschnitt 72: Mit dem Raspberry Pi verbinden

Außerdem wird noch das System-Passwort benötigt. Dieses Passwort lautet standardmäßig *raspberry*.

* + 1. Java Update

Um Java zu aktualisieren, ist es hilfreich, die bisher installierte Version des Java Development Kits (JDK) abzufragen. Dies geschieht am Raspberry Pi mit folgendem Befehl. In diesem Fall war Java 7 installiert.

|  |
| --- |
| java -version |

Codeabschnitt 73: Java-Version abfragen

Um eine neue Version von Java auf dem Raspberry Pi installieren zu können, muss zuerst OpenJDK gelöscht werden.

|  |
| --- |
| sudo apt-get purge openjdk\* |

Codeabschnitt 74: Löschen von OpenJDK

Außerdem wird noch der Digital-Key benötigt. Dieser muss auch hinzugefügt werden.

|  |
| --- |
| sudo apt-key adv --recv-key --keyserver keyserver.ubuntu.com EEA14886 |

Codeabschnitt 75: Hinzufügen des Digital-Keys

Folglich wird die Paket-Quelle festgelegt. Dafür sind mehr als nur ein einziger Befehl notwendig.

|  |
| --- |
| sudo vim /etc/apt/sources.list  deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main  deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main |

Codeabschnitt 76: Hinzufügen des Digital-Keys

Das neue JDK kann jetzt installiert und das alte deinstalliert werden. Das wahrscheinlich stabilste System ist nach wie vor Java 8.

|  |
| --- |
| sudo apt-get update  sudo apt-get install oracle-java8-installer  sudo apt-get install oracle-java8-set-default  sudo apt-get purge openjdk\*  sudo apt-get purge java7\*  sudo apt-get autoremove |

Codeabschnitt 77: Installation von Java 8

Um den Erfolg unserer Aktualisierung festzustellen, kann wieder mit folgendem Befehl die Java-Version überprüft werden:

|  |
| --- |
| java -version |

Codeabschnitt 78: Java-Version abfragen

* + 1. Implementierung von Pi4J

Die Pi4J-Bibliotheken werden für die serielle Übertragung zwischen Arduino und Raspberry Pi benötigt. Für die Installation müssen die JAR-Bibliotheken von *pi4j.com* heruntergeladen werden. Aus dieser Library werden folgende drei JAR-Dateien benötigt:

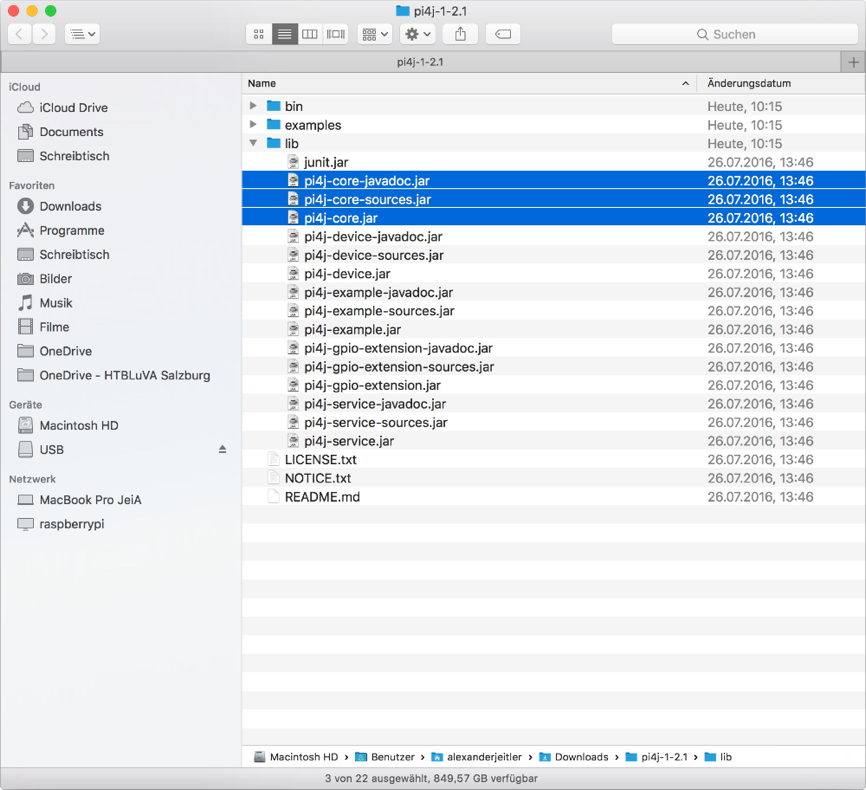


Abbildung 35: Pi4J Bibliothek

Diese JAR-Dateien können unserem Projekt unter *Project Properties* in Netbeans hinzugefügt werden. Damit Pi4J auf dem Pi funktioniert, ist neben den Bibliotheken ein Installationsvorgang notwendig.

|  |
| --- |
| curl -s get.pi4j.com | sudo bash |

Codeabschnitt 79: Installation von Pi4J

Es kann sein, dass während der Installation ein Fehler auftritt. Dieser ist möglicherweise auf eine ungültige Kernel-Version zurückzuführen. Um eine fehlerfreie Installation von Pi4J gewährleisten zu können, ist ein Kernel-Downgrade notwendig. Dafür müssen folgende Befehle in der Kommandozeile ausgeführt werden:

|  |
| --- |
| sudo apt-get update  sudo apt-get dist-upgrade  sudo rpi-update 52241088c1da59a359110d39c1875cda56496764 |

Codeabschnitt 80: Kernel Downgrade

Die Installation von Pi4J sollte jetzt problemlos möglich sein.

* + 1. Remote Platform

Um auf dem Computer remote programmieren zu können, muss eine Remote-Plattform hinzugefügt werden. Dies geschieht innerhalb der Registerkarte *Tools/Java Platforms* mit einem Klick auf den Button *Add Platform*.

|  |  |  |
| --- | --- | --- |
|  |  | |
| Abbildung 36: Java Platform Manager | | |
| Im sich öffnenden Fenster wählt man selbstverständlich *Remote Java Standard Edition*. Mit einem Klick auf *Next* öffnet sich nun ein Menüfenster, wo einige Parameter einzustellen sind. | |  |
|  | | Abbildung 37: Hinzufügen einer Java Platform |
|  | | Platform Name: Name der Plattform  Host: IP-Adresse des Raspberry Pis  Remote JRE Path: An welchem Pfad ist die installierte JRE? (PI) */usr/lib/jvm/jdk-8-oracle-arm32-vfp-hflt/jre/*  Working Dir: An welchem Pfad sollen die Programme gespeichert werden? (PI) */home/pi/NetBeansProjects/* |
| Abbildung 38: Hinzufügen einer Java Platform | |  |

Wenn alle Einstellungen gesetzt wurden, muss auf Finish gedrückt werden. Damit die Programme auch wirklich ausgeführt werden können, muss man Vollzugriffs-Rechte vergeben. Das Kürzel *sudo* ist bei Unix-Betriebssystemen dafür maßgeblich. Das Akronym sudo bedeutet so viel wie „Super-DO“. Um *sudo* wirksam zu machen, muss in den *Java Platform Manager* gewechselt werden und das *Execution Prefix* gesetzt werden.

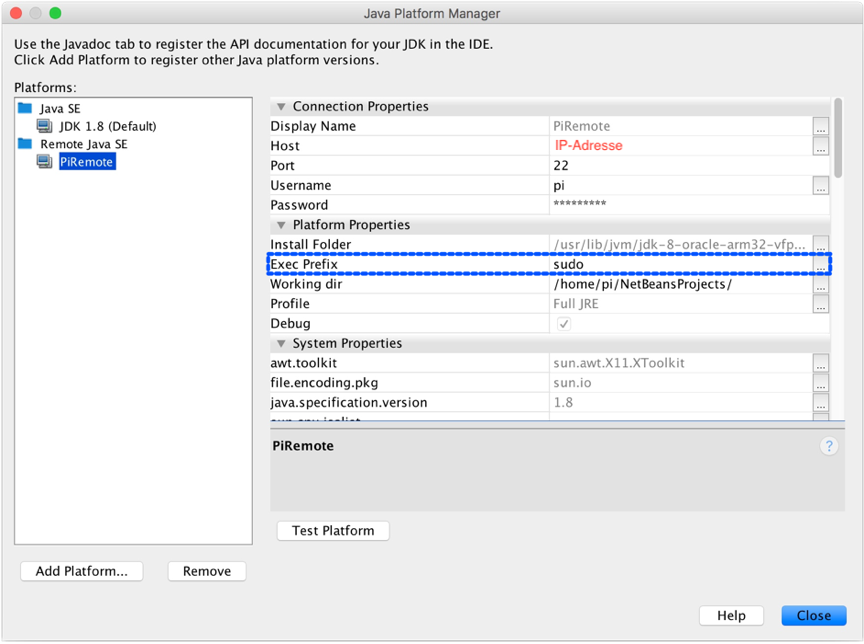


Abbildung 39: Java Platform Manager

Bei jedem neu erstellten Projekt muss auch die *Remote-Plattform* als Ausführungsplattform ausgewählt werden. Dazu wird auf das Projekt rechts-geklickt und *Properties* gewählt.

|  |  |
| --- | --- |
|  |  |
| Abbildung 40: Project Properties | |

Unter dem Reiter *Run* muss nun bei *Runtime Platform* unsere Plattform ausgewählt werden.

* + 1. Modifizierung der build.xml

Auf dem Pi können verschiedenste Programme Remote (sogar Pi4J) ausgeführt werden. Bedingung dafür ist das Einstellen der richtigen IP-Adresse. Leider gibt es aber noch ein Problem mit dem Ausführen von Programmen mit grafischen Benutzeroberflächen. Um dies trotzdem zu ermöglichen, muss die Datei *build.xml* modifiziert werden. Die Datei wird bei jedem Anlegen eines neuen Projektes von Netbeans generiert. Sie lässt sich im Stammverzeichnis des Projekts finden.

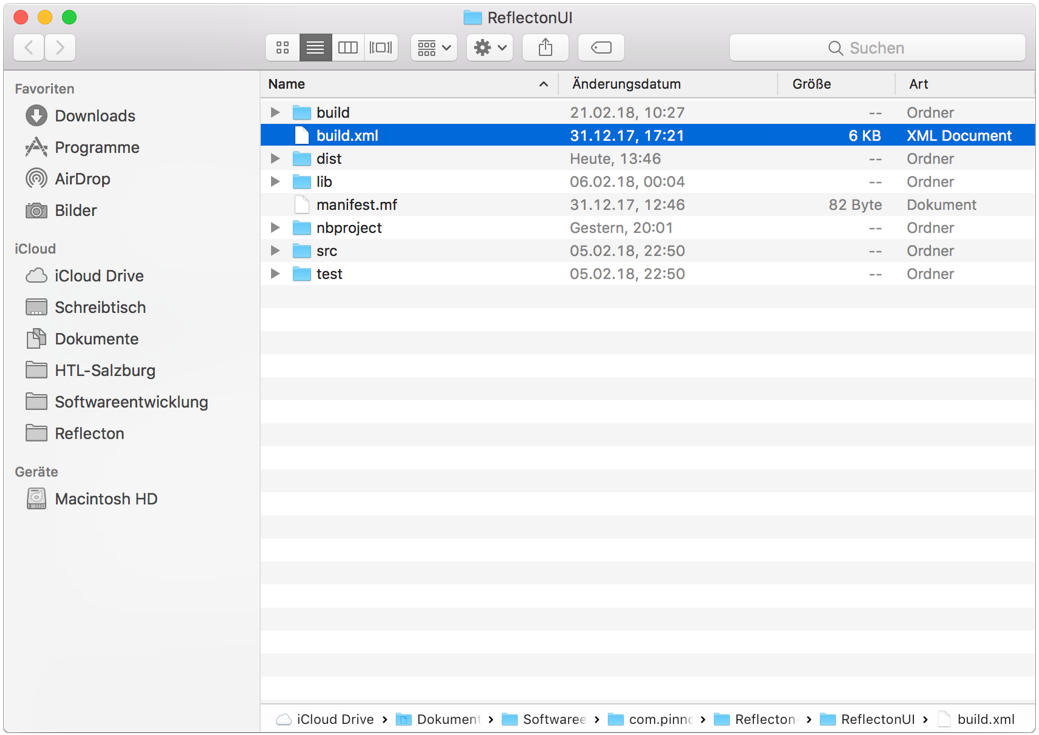


Abbildung 41: Hauptverzeichnis

Es muss die Datei so abgeändert werden, dass die Umgebungsvariable *DISPLAY* gesetzt wird, um den X11-Server damit zu konfigurieren.

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**project name="ReflectonUI" default="default" basedir="."**>  <**description**>Builds, tests, and runs the project ReflectonUI.</**description**>  <**import file="nbproject/build-impl.xml"**/>  <**target name="-copy-to-remote-platform"**>  <**macrodef name="runwithpasswd" uri="http://www.netbeans.org/..."**>  <**attribute name="additionaljvmargs" default=""**/>  <**sequential**>  <**sshexec ...** />  <**scp ...** </**scp**>  <**antcall target="profile-rp-calibrate-passwd"**/>  <**sshexec host="${remote.platform.host}"  port="${remote.platform.port}"  username="${remote.platform.user}"   password="${remote.platform.password}"  trust="true"  usepty="true"  command="export DISPLAY=:0; cd '${remote.project.dir}';   ${remote.platform.exec.prefix}  ' ${remote.java.executable}' @{additionaljvmargs} -  Dfile.encoding=${runtime.encoding} ${run.jvmargs}   ${run.jvmargs.ide} -jar ${remote.dist.jar}   ${application.args}"**/>  </**sequential**>  </**macrodef**>  <**macrodef name="runwithkey" uri="http://www.netbeans.org/..."**>  <**attribute name="additionaljvmargs" default=""**/>  <**sequential**>  <**fail ...** </**fail**>  <**sshexec ...** />  <**scp ...** </**scp**>  <**antcall target="profile-rp-calibrate-key"**/>  <**sshexec host="${remote.platform.host}"  port="${remote.platform.port}"   username="${remote.platform.user}"   keyfile="${remote.platform.keyfile}"   passphrase="${remote.platform.passphrase}"  trust="true" usepty="true"  command="export DISPLAY=:0; cd '${remote.project.dir}';   ${remote.platform.exec.prefix}'  ${remote.java.executable}' @{additionaljvmargs} -  Dfile.encoding=${runtime.encoding} ${run.jvmargs}   ${run.jvmargs.ide} -jar ${remote.dist.jar}   ${application.args}"**/>  </**sequential**>  </**macrodef**>  </**target**> </**project**> |

Codeabschnitt 81: Konfigurieren des X11-Servers (*build.xml*)

Im oben abgebildeten XML-Code wurden die zu ändernden Teile mit gelber Farbe hinterlegt. Wie man erkennen kann, wurde die Umgebungsvariable *DISPLAY* mit dem Wert 0 belegt. Um nun am Raspberry Pi GUI-Anwendungen remote ausführen zu können, müssen bei jedem Neustart zwei Kommandos eingegeben werden.

|  |
| --- |
| xhost +local:all  unset DISPLAY |

Codeabschnitt 82: Voreinstellung zum Ausführen von GUI-Apps

Die zwei Kommandos können auch mithilfe eines Python-Skripts, welches bei jedem Neustart automatisch ausgeführt wird, abgehandelt werden.

* + 1. Bildschirm rotieren

Da der Spiegel hochkant aufgestellt wird, muss der Bildschirm rotiert werden. Dies geschieht mit einer einfachen Änderung der Datei */boot/config.txt*. Die Datei muss beispielsweise mit dem *Nano*-Editor aufgerufen werden.

|  |
| --- |
| sudo nano /boot/config.txt |

Codeabschnitt 83: Rotieren des Bildschirms (*/boot/config.txt*)

Die Änderung des Parameters von *display\_rotate=0* auf *display\_rotate=3*, rotiert den Bildschirm in die gewünschte Richtung.

* 1. **Hinzufügen der Bibliotheken**

Im Hauptverzeichnis des Projektes ist ein *lib*-Ordner anzulegen. In ihn speichern wir sämtliche Bibliotheken, welche für die Funktionalität des Reflecton-UI-Programmes benötigt werden.

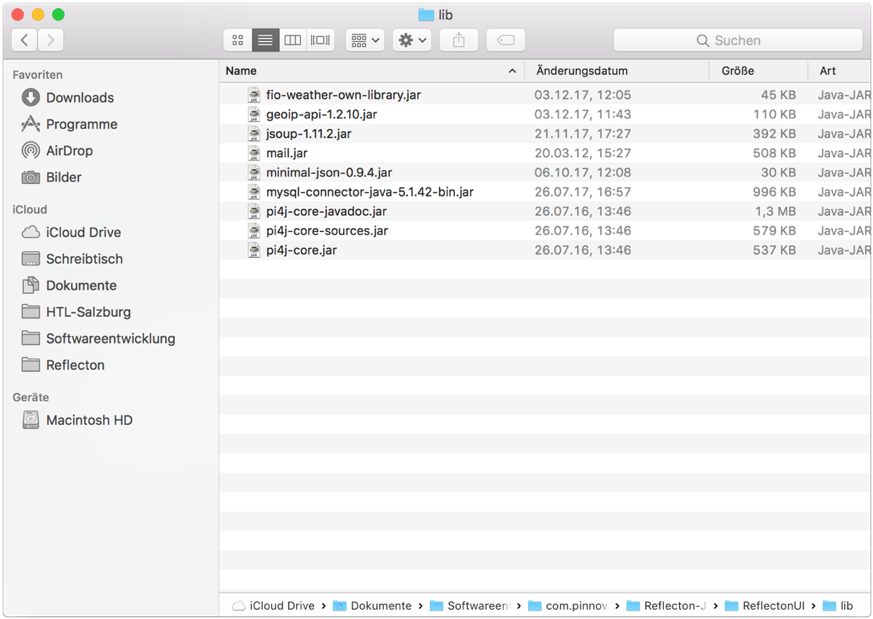


Abbildung 42: Bibliotheken-Ordner

Beim Start des UI-Programmes muss zuerst die komplette Widget-Anordnung heruntergeladen werden. Dabei hilft die *mysql-connector-java-5.1.42-bin.jar*-Library.[[4]](#footnote-5) Für die serielle Übertragung via Pi4J müssen die drei JAR-Dateien *pi4j-core-javadoc.jar*, *pi4j-core-sources.jar* und *pi4j-core.jar* (*pi4j.com*) vorhanden sein.[[5]](#footnote-6) Außerdem wird die Bibliothek *fio-weather-own-library.jar* benötigt.[[6]](#footnote-7) Diese wurde von uns aus mehreren Java-Files von David Ervideira (Github: dvdme) zusammengestellt. Für die Wetterabfrage wird außerdem noch ein Dienst benötigt, welcher über die IP-Adresse Breiten- und Längengrad herausfinden kann. Für die Positionsabfrage ist die *geoip-api-1.2.10.jar*-Bibliothek verantwortlich.[[7]](#footnote-8) Um E-Mails von einem speziellen Mail-Server herunterladen zu können, wird die *mail.jar*-Bibliothek benötigt.[[8]](#footnote-9) Neben den zuletzt genannten Libraries gibt es noch die *jsoup-1.11.2.jar*-Bibliothek[[9]](#footnote-10), sowie die *minimal-json-0.9.4.jar*-Bibliothek[[10]](#footnote-11). Diese werden in verschiedensten Anwendungsformen im Reflecton-UI-Programm benötigt.

Nachdem alle benötigten Libraries, wie in der obigen Darstellung abgebildet, in den *lib*-Ordner kopiert wurden, müssen diese noch dem Java-Programm hinzugefügt werden. Dazu wird auf das Projekt rechts-geklickt und *Properties* gewählt.

Mit einem Klick auf *Add JAR/Folder* kann zum Pfad unseres *lib*-Ordners navigiert werden. Danach sind alle JAR-Dateien auszuwählen und schlussendlich hinzuzufügen.

|  |  |
| --- | --- |
|  |  |
| Abbildung 43: Project Properties | |

Nachdem sich alle gewünschten Bibliotheken in der weißen Box befinden, wird das Fenster mit *OK* geschlossen.

* 1. **Konfigurationsdateien**

Im *config*-Ordner befinden sich vier Java-Konfigurationsdateien mit den Namen *DatabaseConfig.java*, *FileConfig.java*, *FontConfig.java*, sowie *ImageConfig.java*. Diese Konfigurationsdateien sind zur Speicherung von programmrelevanten Daten verantwortlich.

* + 1. ReflectonUI/com/pinnovations/config/DatabaseConfig.java

|  |
| --- |
| **public class** DatabaseConfig {**public static final** String ipAddress = **"173.212.224.225"**;  **public static final** String port = **"3306"**;  **public static final** String user = **"reflecton"**;  **public static final** String password = **"\*\*censored\*\*"**;  **public static final** String databaseName = **"reflecton"**;  **public static final** String tableName = **"user"**;  **public static** Column[] columnElements = {  **new** Column(**"first\_name"**, **"varchar(255)"**),  **new** Column(**"last\_name"**, **"varchar(255)"**),  **new** Column(**"email"**, **"varchar(255)"**),  **new** Column(**"password"**, **"varchar(255)"**),  **new** Column(**"data"**, **"varchar(255)"**)  }; } |

Codeabschnitt 84: Konfigurationsdatei der Datenbank (*DatabaseConfig.java*)

Die Datei *DatabaseConfig.java* beinhaltet diverse Datenbankservereinstellungen, sowie einen Abgleich der Struktur der Tabelle in Form eines Arrays. Neben der IP-Adresse des zugehörigen Servers, dem Benutzernamen der SQL-Datenbank mit zugehörigem Passwort, dem Namen der Datenbank, sowie dem Namen der Tabelle, findet sich ein Array Namens *columnElements* wieder, welches die Struktur der Tabelle abbildet.

* + 1. ReflectonUI/com/pinnovations/config/FileConfig.java

|  |
| --- |
| **public class** FileConfig { **public static final** String reflectonEmailPath = **"/home/pi/Documents"**  + **"/Reflecton/Email.txt"**; } |

Codeabschnitt 85: Konfigurationsdatei der Dateienpfade (*FileConfig.java*)

Die obige Konfigurationsdatei *FileConfig.java* ist nur zum Abspeichern des Pfades der *Email.txt*-Datei verantwortlich. Dies ist jene Datei, die vom PHP-Interface, welches am Apache-Server des Raspberry Pis läuft, bei Smartphone-Aufruf erstellt wird.

* + 1. ReflectonUI/com/pinnovations/config/FontConfig.java

|  |
| --- |
| **public class** FontConfig {  **public static final** String pathRobotoRegular = **"/fonts/Roboto-Regular.ttf"**;  ... } |

Codeabschnitt 86: Konfigurationsdatei der Schriftart (*FontConfig.java*)

Da eine benutzerdefinierte Schriftart im TTF-Format für die Anzeige verwendet wird, ist es hilfreich eine Datei zu haben, welche Pfade innerhalb des Projekts abspeichert. Diese heißt in diesem Fall *FontConfig.java*.

* + 1. ReflectonUI/com/pinnovations/config/ImageConfig.java

|  |
| --- |
| **public class** ImageConfig {  **public static final** String pathEmailWidget = **"/assets/EmailWidget.png"**;  ... } |

Codeabschnitt 87: Konfigurationsdatei der Bilder (*ImageConfig.java*)

Die *ImageConfig.java*-Datei hat projektlokale Pfade zu den darzustellenden Bildern gespeichert. Die meisten dieser Bilder werden zur Darstellung der aktuellen Wetterverhältnisse verwendet. Nebenbei gibt es aber auch noch Design-Muster für das Email-, sowie das Kalender-Widget.

* 1. **Datenbankspezifische Dateien**

Der *database*-Ordner beinhaltet die Dateien *Column.java*, *Database.java* und *InterfaceDatabase.java*. Während *Column* und *Database* Klassen sind, von jenen in anderen Dateien Instanzen erstellt werden, ist die *InterfaceDatabase* eine Klasse, welche nur über eine einzige statische Funktion verfügt.

* + 1. ReflectonUI/com/pinnovations/database/Column.java

|  |
| --- |
| **public class** Column {  **private** String name;  **private** String type;    **public** Column(String name, String type) {  **this**.name = name;  **this**.type = type;  }  **public** String getName() { **return this**.name; }  **public** String getType() { **return this**.type; }  **public** String toString() { **return this**.name + **" "** + **this**.type; } } |

Codeabschnitt 88: Klasse des Tabellenschemas (*Column.java*)

Die Klasse *Column* ist eine einfache Hilfsklasse, welche das Schema einer Datenbanktabelle abbildet. Es wird hier also ein neuer Datentyp etabliert, welcher eine Tabellenstruktur speichern kann. Dies lässt sich auch aus der Datei *DatabaseConfig.java* unschwer erkennen, da hier ein Array aus Objekten der *Column*-Klasse instanziiert wurde. In der *Column*-Klasse sind die zwei einzigen Funktionen sogenannte „Getter“-, sowie „Setter“-Methoden.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| name | Name des Feldes in der Datenbank z.B. 'email' |
| type | Datentyp des Feldes z.B. 'varchar(255)' |

* + 1. ReflectonUI/com/pinnovations/database/Database.java

Die Klasse *Database* ist für die Interaktion mit der Datenbank zuständig. In dieser Klasse brauchen wir allerdings nur eine Abhandlung eines SQL-Statements. Diese befindet sich in der Funktion *selectRow(...)*. Die Funktionen werden später einzeln erklärt, da der Code sonst zu groß und somit zu unhandlich wäre.

Membervariablen

Im Programmcode ist zu erkennen, dass die Membervariablen im Konstruktor mit Daten initialisiert werden. Diese Daten kommen ursprünglich aus den Konfigurationsdateien.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| driver | Diese Membervariable stellt den Treiber von JDBC in einem String dar. JDBC ist kurz erklärt eine Datenbankschnittstelle für Java. |
| ipAddress | Unter dieser Membervariable wird eine Zeichenkette verstanden, welche den Host darstellt. Dies kann beispielsweise die IP-Adresse '173.212.224.225', oder auch der String 'localhost' sein. |
| port | Die folgende Membervariable stellt den Port am Server dar, auf jenem der SQL-Server läuft. Das MySQL Database System hat prinzipiell den Port '3306'. |
| url | Die URL, welche für die Verbindungsanfrage notwendig ist, wird in der Funktion *connectToDatabase()* aus verschiedenen Parametern zusammengebaut. |
| user | Der Benutzername ist zur Anmeldung an einer SQL-Datenbank notwendig. Durch ihn ist eine Absicherung der Datenbank gegeben. Außerdem ist jedem Benutzer eine gewisse Anzahl an Rechten zugeschrieben. So ist es z.B. nicht schlau den Benutzer 'root' zu verwenden, da im PHP-Skript mit diesem alles gemacht werden kann. |
| password | Um eine erfolgreiche Authentifizierung am Datenbankserver zu erzielen, wird zugehörig zum Benutzernamen selbstverständlich auch ein Passwort benötigt. |
| databaseName | Diese Membervariable enthält den Namen der Datenbank am Server. |
| tableName | Der Name der Tabelle wird in dieser Membervariable gespeichert. |
| columnElements | Diese Membervariable stellt die Spaltennamen der Tabelle dar. Die Daten kommen von der *DatabaseConfig.java*-Datei. |
| $databaseConnection | Um eine Verbindung zur Datenbank aufzubauen und jene auch noch zu steuern, wird diese Membervariable unumgänglich benötigt. Die Initialisierung der Variable erfolgt durch die Verbindungsanfrage in der Funktion *connectToDatabase()*. |

Funktionen

Als Funktionen bzw. Methoden gibt es den bereits erwähnten Custom-Konstruktor  
*Database(...),* sowie die Funktionen *connectToDatabase()*, *closeConnectionToDatabase()* und *selectRow(...)*.

Der Konstruktor *Database(...)*, ist für die Initialisierung der meisten Membervariablen verantwortlich.

|  |
| --- |
| **public** Database(String ipAddress, String port, String user, String password,  String databaseName, String tableName, Column[] columnElements) {  **this**.driver = **"com.mysql.jdbc.Driver"**;  **this**.ipAddress = ipAddress;  **this**.port = port;  **this**.user = user;  **this**.password = password;  **this**.databaseName = databaseName;  **this**.tableName = tableName;  **this**.columnElements = columnElements; } |

Codeabschnitt 89: Konstruktor der Datenbankschnittstelle (*Database.java*)

Aus diesem Programmcode geht klar hervor, wie die Membervariablen von den Funktionsparametern initialisiert werden.

Zu Beginn ist ein Verbindungsaufbau mit der Datenbank notwendig. Dafür ist die Funktion *connectToDatabase()* zuständig.

|  |
| --- |
| **public** String connectToDatabase() **throws** Exception {  **this**.url = **"jdbc:mysql://"** + **this**.ipAddress + **":"** + **this**.port+ **"/"** + **this**.databaseName + **"?autoReconnect=true&useSSL=false"**;  Class.forName(driver);  **this**.databaseConnection = DriverManager.getConnection(**this**.url, **this**.user,   **this**.password);  **return this**.url; } |

Codeabschnitt 90: Funktion zum Verbindungsaufbau (*Database.java*)

In diesem Programmcode wird die *url*-Membervariable aus verschiedensten Übergabeparametern zusammengebaut. Mithilfe der Funktion *getConnection(...)* wird die Verbindungsanfrage gestellt. Als Return-Wert wird *url* zurückgegeben.

Um die Verbindung zu beenden wird die Funktion *closeConnectionToDatabase()* verwendet. Es wird stets empfohlen, die Verbindung zur Datenbank zu beenden, da ein Verbindungsabbruch zu Problemen führen kann.

|  |
| --- |
| **public void** closeConnectionToDatabase() **throws** Exception {  **this**.databaseConnection.close(); } |

Codeabschnitt 91: Funktion zum Beenden der Verbindung (*Database.java*)

Wie im Code ersichtlich, wird in unserer Funktion nur eine weitere Funktion namens *close()* aufgerufen. Dies ist eine Funktion des *databaseConnection*-Objekts.

Die Funktion *selectRow(...)* ist der wichtigste Teil der *Database*-Klasse. Sie wird in unserem Reflecton-UI-Programm zum Abrufen des *data*-Feldes benötigt, in welchem die Anordnung der Widgets, sowie die spezifischen Kontodaten gespeichert sind.

|  |
| --- |
| **public** List<String> selectRow(Column[] columnElementsToSelect,  String[] rowElementsToSelect) **throws** Exception {  String sqlQuery = **"SELECT \* FROM "** + **this**.tableName + **" WHERE "**;    **for**(**int** i = 0; i < columnElementsToSelect.length; i++) {  sqlQuery = sqlQuery + columnElementsToSelect[i].getName();  sqlQuery = sqlQuery + **"="** + **"'"** + rowElementsToSelect[i] + **"'"**;    **if**(i < columnElementsToSelect.length - 1) {  sqlQuery = sqlQuery + **" AND "**;  }  }    Statement statement = (Statement) **this**.databaseConnection.createStatement();  ResultSet rs = statement.executeQuery(sqlQuery);    List<String> result = **new** ArrayList<String>();   **while** (rs.next()) {  **for** (**int** i = 0; i < columnElements.length; i++) {  result.add(rs.getString(**this**.columnElements[i].getName()));  }  }  **return** result; } |

Codeabschnitt 92: Funktion zum Selektieren eines Datensatzes (*Database.java*)

In der oberen Programmhälfte wird der *sqlQuery* mithilfe einer for-Schleife zusammengebaut. In den Zeilen danach wird der Query mit den Funktionen *createStatement()* und *executeQuery(...)* schlussendlich ausgeführt. Außerdem wird das Ergebnis in Form einer *ArrayList* an die aufrufende Funktion zurückgegeben.

* + 1. ReflectonUI/com/pinnovations/database/InterfaceDatabase.java

|  |
| --- |
| **public class** InterfaceDatabase {**public static** String getField(String columnElementToSelect,  String rowElementToSelect, String field)  **throws** FieldNotFoundException {  Column[] columnElementsToSelect = {  **new** Column(columnElementToSelect, **"varchar(255)"**)  };  String[] rowElementsToSelect = {rowElementToSelect};  String returnValue = **""**;  **try** {Database db = **new** Database(  DatabaseConfig.ipAddress, DatabaseConfig.port,  DatabaseConfig.user, DatabaseConfig.password,  DatabaseConfig.databaseName, DatabaseConfig.tableName,  DatabaseConfig.columnElements);db.connectToDatabase();  List<String> result = db.selectRow(columnElementsToSelect,   rowElementsToSelect);  String[] resultArray = **new** String[result.size()];  result.toArray(resultArray);  **for** (**int** i = 0; i < resultArray.length; i++) {  **if**(DatabaseConfig.columnElements[i].getName().equals(field)) {  returnValue = resultArray[i];  }  }db.closeConnectionToDatabase();  }  **catch** (Exception e) {  System.out.println(e);  }  **if** (returnValue.equals(**""**)) {  **throw new** FieldNotFoundException(**"Field not found!"**);  }  **return** returnValue;  } } |

Codeabschnitt 93: Funktion zum Laden eines Feldes (*InterfaceDatabase.java*)

Die Datei *InterfaceDatabase* ist Eigner einer einzelnen Funktion namens *getField(...)*. Diese Funktion wird benutzt, um ein Feld aus der Datenbank zu laden. Innerhalb der Funktion wird zuerst ein *Database*-Objekt erstellt. Aus diesem wird dann mit der Funktion *selectRow(...)* ein Datensatz heruntergeladen. Mithilfe von weiteren Funktionen wird aus dem Ergebnis ein einzelnes Feld gewonnen. Das Datenbankfeld, das in unserem Programm die höchste Relevanz hat, ist *data*.

* 1. **Downloadspezifische Dateien**

Der Download-Ordner beinhaltet die Dateien *Appointment.java*, *Clock.java*, *Day.java*, *Email.java*, *GeoLocation.java* und die wichtigste Klasse *InterfaceDatabase.java*. Alle Klassen, abgesehen von InterfaceDatabase, sind nur zur Speicherung von Daten vorhanden.

* + 1. ReflectonUI/com/pinnovations/download/Appointment.java

|  |
| --- |
| **public class** Appointment {  **private** String title, startDate, endDate, location;    **public** Appointment() {  **this**.title = **""**;  **this**.startDate = **""**;  **this**.endDate = **""**;  **this**.location = **""**;  }  **public** Appointment(String title, String startDate, String endDate,  String location) {  **this**.title = title;  **this**.startDate = startDate;  **this**.endDate = endDate;  **this**.location = location;  }  **public void** setTitle(String title) { **this**.title = title; }  **public void** setStartDate(String startDate) { **this**.startDate = startDate; }  **public void** setEndDate(String endDate) { **this**.endDate = endDate; }  **public void** setLocation(String location) { **this**.location = location; }  **public** String getTitle() { **return this**.title; }  **public** String getStartDate() { **return this**.startDate; }  **public** String getEndDate() { **return this**.endDate; }  **public** String getLocation() { **return this**.location; } } |

Codeabschnitt 94: Klasse des Terminschemas (*Appointment.java*)

Die erste Hilfsklasse hört auf den Namen *Appointment* und ist für die Speicherung eines Kalendereintrages bzw. Termins verantwortlich. Neben den unten dargestellten Membervariablen werden nur zahlreiche „Getter“- und „Setter“-Methoden verwendet.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| title | Titel des Kalendereintrages |
| startDate | Datum und Uhrzeit an jenem das Event beginnt |
| endDate | Datum und Uhrzeit an jenem das Event endet |
| location | Die Örtlichkeit in jener das Event stattfindet |

* + 1. ReflectonUI/com/pinnovations/download/Clock.java

|  |
| --- |
| **public class** Clock {  **private** String date, time;    **public** Clock() {  **this**.date = **""**;  }  **public** Clock(String date, String time) {  **this**.date = date;  **this**.time = time;  }  **public void** setDate(String date) { **this**.date = date; }  **public void** setTime(String time) { **this**.time = time; }  **public** String getDate() { **return this**.date; }  **public** String getTime() { **return this**.time; } } |

Codeabschnitt 95: Klasse des Uhrenschemas (*Clock.java*)

Für die Uhrzeit ist die Klasse *Clock* zuständig. Sie enthält die beiden Membervariablen *date* und *time*, welche zur Speicherung der Uhrzeit verantwortlich sind. Neben den Membervariablen sind in der Klasse zahlreiche „Getter“- und „Setter“-Methoden implementiert.

* + 1. ReflectonUI/com/pinnovations/download/Day.java

|  |
| --- |
| **public class** Day {  **private** String day;  **private** String[] fields;    **public** Day() {  **this**.day = **null**;  **this**.fields = **null**;  }  **public** Day(String day, String[] forecastfields) {  **this**.day = day;  **this**.fields = forecastfields;  }    **public void** setDay(String day) { **this**.day = day; }  **public void** setFields(String[] fields) { **this**.fields = fields; }  **public** String getDay() { **return this**.day; }  **public** String[] getFields() { **return this**.fields; }    **public** String toString() {  String returnValue = **this**.day + **": "**;  **for** (**int** i = 0; i < **this**.fields.length; i++) {  returnValue += **"\n"** + **this**.fields[i];  }  **return** returnValue;  }  **public static** String getDayByNumber(**int** iterator) {  String day = **null**;  **switch**(iterator) {  **case** Calendar.MONDAY:  day = **"Monday"**;  **break**;  ...  **default**:  day = **null**;  **break**;  }    **return** day;  } } |

Codeabschnitt 96: Klasse des Tageschemas (*Day.java*)

Die Klasse *Day* ist zum Abspeichern der Wetterbedingungen eines Tages zu verwenden. Innerhalb der Klasse gibt es die Membervariablen *day* und *fields*. Während *day* einfach ein String zur Benennung des Tages ist, ist *fields* ein Array mit gespeicherten Wetterparametern wie z.B. Temperatur. Als Methoden gibt es verschiedene „Getter“- und „Setter“-Methoden, sowie eine Funktion, welche auf den Namen *getDayByNumber(...)* hört. Diese Funktion gibt einen Tag abhängig vom Iterator als String zurück.

* + 1. ReflectonUI/com/pinnovations/download/Email.java

|  |
| --- |
| **public class** Email {  **private long** uid;  **private** String subject, from, content;  **public** Email() {  **this**.uid = 0;  **this**.subject = **null**;  **this**.from = **null**;  **this**.content = **null**;  }  **public** Email(**long** uid, String subject, String from, String content) {  **this**.uid = uid;  **this**.subject = subject;  **this**.from = from;  **this**.content = content;  }  **public void** setUid(**long** uid) { **this**.uid = uid; }  **public void** setSubject(String subject) { **this**.subject = subject; }  **public void** setFrom(String from) { **this**.from = from; }  **public void** setContent(String content) { **this**.content = content; }  **public long** getUid() { **return this**.uid; }  **public** String getSubject() { **return this**.subject; }  **public** String getFrom() { **return this**.from; }  **public** String getContent() { **return this**.content; }  **public** String toString() {  **return this**.subject + **" FROM "** + **this**.from + **": "** + **this**.content;  } } |

Codeabschnitt 97: Klasse des Mailschemas (*Email.java*)

Die Klasse Email speichert die notwendigen Daten einer einzelnen Email. Neben den unten dargestellten Membervariablen werden nur zahlreiche „Getter“- und „Setter“-Methoden verwendet.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| uid | Identifikationsnummer der E-Mail |
| subject | Betreff der E-Mail |
| from | Person von jener die E-Mail abgeschickt wurde |
| content | Inhalt der E-Mail |

* + 1. ReflectonUI/com/pinnovations/download/GeoLocation.java

|  |
| --- |
| **public class** GeoLocation {  **private** String countryCode, countryName, regionNumber, regionName, city,   postalCode, latitude, longitude;  **public** GeoLocation() {  **this**.countryCode = **""**;  **this**.countryName = **""**;  **this**.regionNumber = **""**;  **this**.regionName = **""**;  **this**.city = **""**;  **this**.postalCode = **""**;  **this**.latitude = **""**;  **this**.longitude = **""**;  }  **public** GeoLocation(String countryCode, String countryName, String regionNumber,   String regionName, String city, String postalCode,  String latitude, String longitude) {  **this**.countryCode = countryCode;  **this**.countryName = countryName;  **this**.regionNumber = regionNumber;  **this**.regionName = regionName;  **this**.city = city;  **this**.postalCode = postalCode;  **this**.latitude = latitude;  **this**.longitude = longitude;  }  **public void** setCountryCode(String countryCode) {**this**.countryCode = countryCode;}  **public void** setCountryName(String countryName) {**this**.countryName = countryName;}  **public void** setRegionNumber(String regionNumber){**this**.regionNumber=regionNumber;}  **public void** setRegionName(String regionName) { **this**.regionName = regionName; }  **public void** setCity(String city) { **this**.city = city; }  **public void** setPostalCode(String postalCode) { **this**.postalCode = postalCode; }  **public void** setLatitude(String latitude) { **this**.latitude = latitude; }   **public void** setLongitude(String longitude) { **this**.longitude = longitude; }  **public** String getCountryCode() { **return this**.countryCode; }  **public** String getCountryName() { **return this**.countryName; }  **public** String getRegionNumber() { **return this**.regionNumber; }  **public** String getRegionName() { **return this**.regionName; }  **public** String getCity() { **return this**.city; }  **public** String getPostalCode() { **return this**.postalCode; }  **public** String getLatitude() { **return this**.latitude; }  **public** String getLongitude() { **return this**.longitude; } } |

Codeabschnitt 98: Klasse des Ortschemas (*GeoLocation.java*)

Um alle verschiedenen Ortsparameter in einer Klasse beisammen zu haben, wurde die Klasse *GeoLocation* etabliert. Diese besteht aus „Getter“- und „Setter“-Methoden, sowie verschiedensten Membervariablen.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| countryCode | Landesnummer |
| countryName | Landesname |
| regionNumber | Regionsnummer |
| regionName | Regionsname |
| city | Stadt |
| postalCode | Postleitzahl |
| latitude | Breitengrad |
| longitude | Längengrad |

* + 1. ReflectonUI/com/pinnovations/download/InterfaceDownload.java

Die *InterfaceDownload.java*-Datei ist für die Beschaffung von jeglicher Art von Informationen verantwortlich. Da der circa 500 Zeilen lange Code viel zu unhandlich ist, werden die einzelne Funktionen einzeln erklärt. Außerdem gibt es in diesem statischen Code keine Membervariablen.

Funktionen

In unserer *InterfaceDownload.java*-Datei werden statische Funktionen implementiert. Diese werden nun behandelt.

Die Funktion *getWidgetsArrangement(...)* hat die Aufgabe, die Anordnung der Widgets herunterzuladen. Diese Anordnung ist in der Datenbank gespeichert und folgt dem unten abgebildeten Schema.

|  |
| --- |
| <widget>email:m.mustermann@mail.com:abc</widget>  <widget>calendar:m.mustermann@icloud.com:abc</widget>  <widget>weather</widget>  <widget>clock</widget>" |

Codeabschnitt 99: Widget-Anordnung

Wie am Return-Wert der unten abgebildeten Funktion recht deutlich erkennbar ist, wird die Anordnung der einzelnen Widgets in einem Array gespeichert und schlussendlich zurückgegeben.

|  |
| --- |
| **public static** String[] getWidgetsArrangement(String path) {String column = **"email"**;String reflectonEmail = Filereader.getLines(path)[0];  String returnField = **"data"**; String widgetsArrangement = **""**;  **try** {  widgetsArrangement = InterfaceDatabase.getField(column,  reflectonEmail, returnField);  } **catch** (FieldNotFoundException e) {  widgetsArrangement = **"<widget>email</widget>"**  + **"<widget>calendar</widget>"**  + **"<widget>weather</widget>"**  + **"<widget>clock</widget>"**;  } String widgetsArrangementHelp = widgetsArrangement.replaceAll(**"</widget>"**, **"/"**)  .replaceAll(**"<widget>"**, **""**);String widgetsArrangementString = widgetsArrangementHelp.substring(0,   widgetsArrangementHelp.length() - 1);String[] widgetsArrangementArray = widgetsArrangementString.split(**"/"**);    **return** widgetsArrangementArray; } |

Codeabschnitt 100: Funktion zum Laden der Widget-Anordnung (*InterfaceDownload.java*)

Hinter dem Parameter path steckt der Pfad zur *Email.txt*-Datei, welche vom PHP-Interface am Pi erstellt wird. Im Fortlauf des Codes wird die Schlüsselspalte als *email* festgelegt und das rückzugebende Feld als *data* spezifiziert. Folglich wird die Funktion *getField(...)* aufgerufen. Das *data*-Feld wird danach in mehreren Schritten in Elemente des Arrays *widgetsArrangementArray* zerlegt und jenes wird rückgegeben.

Die E-Mail-Adresse, sowie das Passwort eines bestimmten Benutzers wird in der Datenbank im *data*-Feld abgespeichert. Mit jenen zwei Informationen kann die unten gelistete Funktion aufgerufen werden. Zuerst muss aus dem *email*-String der benötigte IMAP-Server ausgelesen werden. Danach wird nach einigen vorbereitenden Schritten zu diesem Server mithilfe von *store.connect(...)* verbunden.

|  |
| --- |
| **public static** Email[] getEmails(String email, String password, **long** numberOfEmails) **throws** MessagingException {  Email[] emails = **null**;  String server = **null**;    **if**(email.contains(**"@gmail.com"**)) {  server = **"imap.gmail.com"**;  }  **else if**(email.contains(**"@gmx"**)) {  server = **"imap.gmx.net"**;  }  **else if**(email.contains(**"@outlook.com"**)) {  server = **"imap-mail.outlook.com"**;  }  **else if**(email.contains(**"@icloud.com"**)) {  server = **"imap.mail.me.com"**;  }    Properties props = System.getProperties();  props.setProperty(**"mail.store.protocol"**, **"imaps"**);  Session session = Session.getDefaultInstance(props, **null**);  Store store = session.getStore(**"imaps"**);  **try** {  store.connect(server, email, password);  IMAPFolder folder = (IMAPFolder) store.getFolder(**"INBOX"**);  **try** {  **if** (!folder.isOpen()) {  folder.open(Folder.READ\_ONLY);  }  **long** largestUid = folder.getUIDNext() - 1;  **if** (numberOfEmails == -1) {  numberOfEmails = largestUid;  }  Message[] messages = folder.getMessagesByUID(largestUid-numberOfEmails + 1,   largestUid);  FetchProfile metadataProfile = **new** FetchProfile();  metadataProfile.add(FetchProfile.Item.FLAGS);  metadataProfile.add(FetchProfile.Item.ENVELOPE);  folder.fetch(messages, metadataProfile);    emails = **new** Email[messages.length];  **for** (**int** i = 0; i < messages.length; i++) {emails[i] = **new** Email();  emails[i].setUid(folder.getUID(messages[messages.length - i - 1]));  emails[i].setSubject(messages[messages.length - i - 1].getSubject());  emails[i].setFrom(  messages[messages.length - i - 1].getFrom()[0].toString()  );  **try** {  **if** (messages[messages.length - i - 1].isMimeType(**"text/plain"**)) {emails[i].setContent(  messages[messages.length - i - 1].getContent().toString()  );  }   **else if** (messages[messages.length - i - 1]  .isMimeType(**"multipart/\*"**)) {  String result = **""**;  MimeMultipart mimeMultipart = (MimeMultipart)  messages[messages.length - i - 1].getContent();   **int** count = mimeMultipart.getCount();  **for** (**int** j = 0; j < count; j ++) {  BodyPart bodyPart = mimeMultipart.getBodyPart(j);  **if** (bodyPart.isMimeType(**"text/plain"**)){  result = result + **"\n"** + bodyPart.getContent();  **break**;}  **else if** (bodyPart.isMimeType(**"text/html"**)){  String html = (String) bodyPart.getContent();  result = result + **"\n"** + Jsoup.parse(html).text();  }  }emails[i].setContent(result);  }  } **catch**(Exception e) {  e.printStackTrace();  }  }  }  **finally** {  **if** (folder.isOpen()) {  folder.close(**true**);  }  }  }  **finally** {  store.close();  }   **return** emails; } |

Codeabschnitt 101: Funktion zum Laden der E-Mails (*InterfaceDownload.java*)

Innerhalb der for-Schleife wird ein Array des Typs *Email* mit Instanzen mit je einer *UID*-Nummer, einem Betreff, sowie einem Absender befüllt. Danach folgen noch einige Schritte, welche zur ordnungsgemäßen Ausführung der Funktion nötig sind. Das oben genannte Array, welches die *Emails* der Anzahl *numberOfEmails* hat, wird schlussendlich mit dem Return-Wert zurückgegeben.

Um neben Emails auch Kalendereinträge abrufen zu können, wird die Funktion *getAppointments(...)* aufgerufen. Diese Funktion ist zurzeit nur auf iCloud-Kalender spezialisiert, was gleich durch die erste if-Abfrage deutlich wird.

|  |
| --- |
| **public static** Appointment[] getAppointments(String email, String password, String startDate) {  **if**(email.contains(**"@icloud.com"**)) {  Appointment[] appointments = **null**;  String s = **null**;    **try** {File file = getFile(**"/scripts/IcloudCalendar.py"**);  String path = file.getAbsolutePath();  Process p = Runtime.getRuntime().exec(**"python "** + path + **" "** + email + **" "** + password + **" 0 "** + startDate);    String output = **""**;  BufferedReader stdInput = **new** BufferedReader(**new** InputStreamReader(p.getInputStream()));  **while** ((s = stdInput.readLine()) != **null**) {  output += s;  }  appointments = formatOutputAppleCalendar(output);  }  **catch** (IOException e) {  e.printStackTrace();  }    **return** appointments;  }  **else** {  **return null**;  }   **return** appointments; } |

Codeabschnitt 102: Funktion zum Laden der Kalendereinträge (*InterfaceDownload.java*)

Im oben genannten Code wird mithilfe der Funktion *getFile(...)*, welche später noch erklärt wird, ein Skript namens *IcloudCalendar.py* geladen. Jenes Python-Skript wird mithilfe eines Prozesses ausgeführt. Die daraus entstehende Ausgabe wird unter Verwendung eines *BufferedReaders* eingelesen und weiterverarbeitet. Nach einer notwendigen Formatierung mit der Methode *formatOutputAppleCalendar(...)*, wird ein *Appointment*-Array zurückgegeben, welches einzelne Kalendereinträge mit etwaigen Parametern enthält. Das oben genannte Python-Skript *IcloudCalendar.py* wird nun erklärt.

|  |
| --- |
| **import** sys **from** pyicloud **import** PyiCloudService **from** datetime **import** datetime  api = PyiCloudService(str(sys.argv[1]), str(sys.argv[2])) startDate = str(sys.argv[4]) startDateArray = startDate.split(**"-"**)  from\_dt = datetime(int(startDateArray[0]), int(startDateArray[1]), int(startDateArray[2])) **if** startDateArray[1] < 12:  to\_dt = datetime(int(startDateArray[0]), int(startDateArray[1])+1, int(startDateArray[2])) **else**:  to\_dt = datetime(int(startDateArray[0]), 12, 31)  events = api.calendar.events(from\_dt, to\_dt) **print**(events) |

Codeabschnitt 103: Python-Skript zum Laden der Kalendereinträge (*IcloudCalendar.py*)

Bei diesem Skript wurde auf die Verwendung von *pyicloud* zurückgegriffen. Jene benötigte Bibliothek wird mit folgendem Kommando installiert:

|  |
| --- |
| sudo pip install pyicloud |

Codeabschnitt 104: Installieren von *PyIcloud*

Wie im Programmcode des Skripts ersichtlich, müssen zum Erhalten der Kalenderevents mit *api.calendar.events(...)* Startdatum und Enddatum festgelegt werden. Jene zwei Daten sind als Aufrufparameter an das Skript zu übergeben. Neben Start- und Enddatum werden aber noch weitere Parameter wie Emailadresse des iCloud-Accounts sowie Passwort benötigt.

Wie in der zuletzt besprochenen Funktion *getAppointments(...)* ersichtlich,  
wurde eine Methode mit dem Namen *formatOutputAppleCalendar(...)* implementiert. Diese Methode wird zum Formatieren des Outputs des Python-Skripts verwendet.

|  |
| --- |
| **public static** Appointment[] formatOutputAppleCalendar(String output) {  String[] eventArray = output.split(Pattern.quote(**"{"**));  String[][] values;  Appointment[] appointments = **new** Appointment[eventArray.length];  **for** (**int** i = 0; i < eventArray.length; i++) {  String[] help = eventArray[i].split(**", u'"**);  values = **new** String[eventArray.length][help.length];  appointments[i] = **new** Appointment();    **for** (**int** j = 0; j < help.length; j++) {  values[i][j] = help[j];  **if**(values[i][j].contains(**"title"**)) {  ...  }  **else if**(values[i][j].contains(**"startDate"**)) {  ...  }  **else if**(values[i][j].contains(**"endDate"**)) {  ...  }  **else if**(values[i][j].contains(**"location"**)) {  ...  }  }  }  appointments = Arrays.copyOfRange(appointments, 1, appointments.length);  Appointment help;  **for** (**int** i = 0; i < appointments.length - 1; i++) {  **for** (**int** j = i + 1; j < appointments.length; j++) {  **if**(Integer.parseInt(appointments[i].getStartDate().split(**","**)[0])  > Integer.parseInt(appointments[j].getStartDate().split(**","**)[0])) {  help = appointments[i];  appointments[i] = appointments[j];  appointments[j] = help;  }  }  }  **return** appointments; } |

Codeabschnitt 105: Funktion zum Formatieren der Termine (*InterfaceDownload.java*)

Innerhalb des oben dargestellten Blocks wird ein *Appointment*-Array geformt und sortiert. Das fertige Array wird schlussendlich an die aufrufende Funktion zurückgegeben.

Für den Wetterdienst muss der Standort mithilfe der zwei Funktionen *getLocation()* und *getIP()* herausgefunden werden. Dazu wird die Datei GeoLiteCity.dat benötigt.

|  |
| --- |
| **public static** GeoLocation getLocation() {  String ipAddress = getIP();  File file = getFile(**"/files/GeoLiteCity.dat"**);  GeoLocation geoLocation = **new** GeoLocation();  **try** {  LookupService lookup = **new** LookupService(file,   LookupService.GEOIP\_MEMORY\_CACHE);  Location locationServices = lookup.getLocation(ipAddress);    geoLocation.setCountryCode(locationServices.countryCode);  geoLocation.setCountryName(locationServices.countryName);  geoLocation.setRegionNumber(locationServices.region);  geoLocation.setRegionName(regionName.regionNameByCode(  locationServices.countryCode,   locationServices.region));  geoLocation.setCity(locationServices.city);  geoLocation.setPostalCode(locationServices.postalCode);  geoLocation.setLatitude(String.valueOf(locationServices.latitude));  geoLocation.setLongitude(String.valueOf(locationServices.longitude));  }  **catch**(IOException e) {  System.err.println(e.getMessage());  }  **return** geoLocation; }  **public static** String getIP() {  String ipAddress = **""**;  **try** {  URL url = **new** URL(**"http://checkip.amazonaws.com"**);  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(url.openStream()));  ipAddress = br.readLine();  } **catch** (Exception e) {  e.printStackTrace();  }  **return** ipAddress; } |

Codeabschnitt 106: Funktion zum Laden des Ortes (*InterfaceDownload.java*)

Nachdem die IP-Adresse mit dem Aufruf von *getIP()* herausgefunden wurde, wird in der Funktion *geoLocation()* ein *GeoLocation*-Array erstellt, welches sämtliche Parameter der Örtlichkeit enthält.

Wenn Längengrad und Breitengrad ermittelt worden sind, kann das aktuelle Wetter mit *getCurrentWeather(...)*, sowie die Wettervorhersage als Mehrtagesprognose mit *getWeather(...)* geladen werden. Diese Wetterdaten werden schlussendlich an der Benutzeroberfläche angezeigt.

|  |
| --- |
| **public static** Day getCurrentWeather(String latitude, String longitude) {  String apikey = **"7a4a8398d4ae0d6134ea0a4eafcf2b45"**;  ForecastIO fio = **new** ForecastIO(apikey);  fio.setUnits(ForecastIO.UNITS\_AUTO);  fio.setLang(ForecastIO.LANG\_ENGLISH);  fio.getForecast(latitude, longitude);  FIOCurrently currently = **new** FIOCurrently(fio);  String[] help = currently.get().getFieldsArray();  String[] fields = **new** String[help.length];  Day day = **new** Day();  **for** (**int** i = 0; i < help.length; i++) {  fields[i] = help[i] + **": "** + currently.get().getByKey(help[i]);  }  day.setDay(**"Today"**);  day.setFields(fields);  **return** day; } **public static** Day[] getWeather(String latitude, String longitude) {  String apikey = **"7a4a8398d4ae0d6134ea0a4eafcf2b45"**;  ForecastIO fio = **new** ForecastIO(apikey);  fio.setUnits(ForecastIO.UNITS\_AUTO);  fio.setLang(ForecastIO.LANG\_ENGLISH);  fio.getForecast(latitude, longitude);  Calendar calendar = Calendar.getInstance();  FIODaily daily = **new** FIODaily(fio);  Day[] day = **new** Day[daily.days() - calendar.get(Calendar.DAY\_OF\_WEEK)];  **for** (**int** i = 0; i < day.length; i++) {  day[i] = **new** Day();  day[i].setDay(Day.getDayByNumber(i + calendar.get(Calendar.DAY\_OF\_WEEK)));  String[] help1 = daily.getDay(i+calendar.get(Calendar.DAY\_OF\_WEEK))  .getFieldsArray();  String[] help2 = **new** String[help1.length];**for** (**int** j = 0; j < help1.length; j++) {  help2[j] = help1[j] + **": "** + daily.getDay(i + calendar  .get(Calendar.DAY\_OF\_WEEK))  .getByKey(help1[j]);}  day[i].setFields(help2);  }  **return** day; } |

Codeabschnitt 107: Funktion zum Laden des Wetters (*InterfaceDownload.java*)

Die zwei abgebildeten Funktionen sind von der Grundstruktur ziemlich ident. Bei *getCurrentWeather(...)* wird ein *Day*-Objekt mithilfe der FIO-Bibliothek erzeugt und jenes wird zurückgegeben. Im Gegensatz dazu wird bei *getWeather(...)* ein komplexeres *Day*-Array innerhalb einer for-Schleife erstellt. Dieses wird dann schlussendlich als Return-Wert zurückgegeben.

Das wahrscheinlich unkomplizierteste Widget ist die Darstellung einer Uhr. Um dies zu ermöglichen wird die Funktion *getClock()* benötigt.

|  |
| --- |
| **public static** Clock getClock() {  SimpleDateFormat sdfClock = **new** SimpleDateFormat(**"HH:mm dd-MM"**);  Date now = **new** Date();  String strClock = sdfClock.format(now);    String[] clockArray = strClock.split(**" "**);  Clock clk;  **if**(clockArray.length > 1) {  clk = **new** Clock(clockArray[1], clockArray[0]);  }  **else** {  clk = **new** Clock();  }  **return** clk; } |

Codeabschnitt 108: Funktion zum Laden der Uhrzeit (*InterfaceDownload.java*)

Diese Funktion gibt ein Objekt vom Typ *Clock* zurück, welches das aktuelle Datum, sowie die Uhrzeit gespeichert hat.

* 1. **Hilfsdateien**

Die folgenden Dateien sind in mehreren Java-Packages beheimatet. Sie sind keine Elemente des Programmes, die eine zentrale Funktion darstellen, sondern lediglich unterstützende Klassen bzw. Methoden.

* + 1. ReflectonUI/com/pinnovations/filereader/Filereader.java

|  |
| --- |
| **public class** Filereader {  **public static** File getFile(String resource) {  File file = **null**;  java.net.URL res = Filereader.**class**.getResource(resource);  **if** (res.toString().startsWith(**"jar:"**)) {  **try** {  InputStream input = Filereader.**class**.getResourceAsStream(resource);  file = File.createTempFile(**"tempfile"**, **".tmp"**);  OutputStream out = **new** FileOutputStream(file);  **int** read;  **byte**[] bytes = **new byte**[1024];    **while** ((read = input.read(bytes)) != -1) {  out.write(bytes, 0, read);  }  file.deleteOnExit();  } **catch** (IOException ex) {  }  }  **else** {file = **new** File(res.getFile());  }  **if** (file != **null** && !file.exists()) {  **throw new** RuntimeException(**"Error: File "** + file + **" not found!"**);  }  **return** file;  }  **public static** String[] getLines(String path) {  List<String> list = **new** ArrayList<>();  **try** (BufferedReader br = Files.newBufferedReader(Paths.get(path))) {list = br.lines().collect(Collectors.toList());  }  **catch** (IOException e) {  e.printStackTrace();  }  String[] listArray = **new** String[list.size()];  list.toArray(listArray);  **return** listArray;  } } |

Codeabschnitt 109: Funktion zum Auslesen eines Files (*Filereader.java*)

Innerhalb des UI-Programmes werden des Öfteren externe Dateien benötigt. Diese können mithilfe der Funktion *getFile(...)* eingelesen werden. An die Methode muss der Pfad der gewünschten Datei übergeben werden. Wird die Datei gefunden, so generiert die gelistete Funktion ein temporäres File mit demselben Inhalt. Diese Temporär-Datei wird dann schlussendlich an die aufrufende Methode zurückgegeben. Im Fehlerfall hingegen wird ein String ausgegeben, welcher besagt, dass das File nicht gefunden wurde.

Um den Inhalt einer beliebigen Textdatei auch lesen zu können, wird die Funktion *getLines(...)* benötigt. Jene Funktion liest den Inhalt einer Datei ein und speichert ihn in einem String-Array.

* + 1. ReflectonUI/com/pinnovations/support/FontSupport.java

|  |
| --- |
| **public class** FontSupport {  **public static** Font getFont(String path, **float** size) {  **try** {  Font font = Font.createFont(Font.TRUETYPE\_FONT, getFile(path));  **return** font.deriveFont(size);  }   **catch** (IOException|FontFormatException e) {  }  **return null**;  } } |

Codeabschnitt 110: Funktion zum Auslesen der Schriftart (*FontSupport.java*)

Die oben abgebildete Methode dient zum Auswählen einer bestimmten Schriftart. Dabei muss die Truetype-Font-Datei (.ttf) als Pfad übergeben werden. Wenn die Schriftart gefunden wurde, dann wird sie als Return-Wert an die aufrufende Funktion zurückgeben.

* + 1. ReflectonUI/com/pinnovations/support/TableSupport.java

Aufgrund ihrer Größe werden die Funktionen der *TableSupport.java*-Datei  
einzeln erklärt. Neben den öffentlichen Funktionen *getFormattedText(...)* und *getFormattedTextWithHeading(...)* befindet sich auch noch die Methode *getTableCellRenderer(...)* in dem File. Da sich die ersten zwei Funktionen sehr ähneln, wird nur *getFormattedText(...)* besprochen.

Funktionen

Die Methode *getFormattedText(...)* generiert eine formatierte Zeichenkette im HTML-Format, welche an einen Text in einer bestimmten Tabelle angepasst ist.

|  |
| --- |
| **public static** String getFormattedText(String text, Font font, **int** columnWidth) {  **int** textWidth;**try** {  textWidth = (**int**)(font.getStringBounds(text,  **new** FontRenderContext(**new** AffineTransform(), **true**, **true**))  .getWidth());  } **catch**(Exception e) {  textWidth = 0;  }**if**(textWidth > columnWidth) {String substring = **""**;  **int** substringWidth = 0;  **int** index = 0;  **while**(substringWidth < columnWidth) {  substring = text.substring(0, index);  substringWidth = (**int**)(font.getStringBounds(substring,  **new** FontRenderContext(**new** AffineTransform(), **true**,   **true**)).getWidth());  index++;  }**if**(index > 2) {index -= 2;**if**(text.contains(**" "**)) {  **while**(text.charAt(index) != **' '**) {  index--;  }  }String firstLine = text.substring(0, index);  String lastLine = text.substring(index) + **" "**; **int** leftTextWidth = (**int**)(font.getStringBounds(lastLine,  **new** FontRenderContext(**new** AffineTransform(),  **true**, **true**)).getWidth());**int** lastLineWidth = 0;  **do** {  lastLineWidth = (**int**)(font.getStringBounds(lastLine,   **new** FontRenderContext(**new** AffineTransform(),  **true**, **true**)).getWidth());  lastLine = lastLine.substring(0, lastLine.length() - 1);  } **while**(lastLineWidth > columnWidth);**if**(leftTextWidth > columnWidth) {lastLine = lastLine.substring(0, lastLine.length() - 1);**if**(lastLine.charAt(lastLine.length() - 1) == **' '**) {  lastLine = lastLine.substring(0, lastLine.length() - 1);  } lastLine += **"..."**;  }**return "<html>"** + firstLine + **"<br>"** + lastLine + **"</html>"**;  }  **else** {**return ""**;  }  }  **else** {**return "<html>"** + text + **"</html>"**;  } } |

Codeabschnitt 111: Funktion zum Formatieren des Textes (*TableSupport.java*)

In der äußeren if-Abfrage lässt sich erkennen, dass zuerst überprüft wird, ob die Breite des Textes größer ist als die Breite der Spalte. Ist das so, dann wird die Zeichenkette aufgespalten und durch einen Zeilenumbruch geteilt.

Die Standardklasse von individuellen Zellen hört in Java auf den Namen *DefaultTableCellRenderer*. Die Funktion *getTableCellRenderer(...)* gibt bei einer bestimmten Schriftart ein einzigartiges Objekt der Klasse *DefaultTableCellRenderer* zurück.

|  |
| --- |
| **public static** DefaultTableCellRenderer getTableCellRenderer(Font font) {  DefaultTableCellRenderer cellRenderer = **new** DefaultTableCellRenderer() {  **private static final long** serialVersionUID = 1L;  **public** Component getTableCellRendererComponent(JTable table, Object value,   **boolean** isSelected, **boolean** hasFocus, **int** row, **int** column) {  **super**.getTableCellRendererComponent(table, value, isSelected, hasFocus,  row, column);  **this**.setFont(font);  **return this**;  }  };  **return** cellRenderer; } |

Codeabschnitt 112: Funktion die einen TableCellRenderer zurückgibt (*TableSupport.java*)

Nach dem Block, der nach der Instanziierung des DefaultTableCellRenderer-Objekts geöffnet wird, werden spezielle Parameter, wie beispielsweise die Schriftart, gesetzt.

* + 1. ReflectonUI/com/pinnovations/exceptions/FieldNotFoundException.java

|  |
| --- |
| **public class** FieldNotFoundException **extends** Exception {  **private static final long** serialVersionUID = 1L;    **public** FieldNotFoundException(String message) {  **super**(message);  } } |

Codeabschnitt 113: Exception wenn ein Feld nicht gefunden wird (*FieldNotFoundException.java*)

Die einzige eigene Klasse, welche als Exception verwendet wird, hört auf den Namen *FieldNotFoundException*. In ihrem Konstruktor befindet sich ein einfacher Aufruf des Konstruktors der Superklasse *Exception*, welchem ein Melde-String bzw. Error-String übergeben wird.

* 1. **Widgets**

In der Ansicht *MainView*, welche später ausführlich erklärt wird, werden die Widgets *EmailWidget*, *CalendarWidget*, *WeatherWidget* und *ClockWidget* dargestellt. Je nachdem, wie ihre Anordnung in der Datenbank gespeichert ist, entscheidet in welcher der vier Ecken sie dargestellt werden.

* + 1. ReflectonUI/com/pinnovations/widgets/Widget.java

Die Klasse *Widget* erbt von *JPanel* und ist die Superklasse späterer spezifischer Widgets wie z.B. *EmailWidget*. Sie ist somit der Sockel späterer spezieller Widgets.

|  |
| --- |
| **public class** Widget **extends** JPanel {  **private static final long** serialVersionUID = 1L;  **private** JLabel heading;  **private int** position;  **public** Widget() {  **this**.setBackground(**new** Color(0, 0, 0));  **this**.setLayout(**null**);  **this**.heading = **null**;  **this**.position = -1;  }  **public** Widget(String heading, **int** pos) {  **this**.setBackground(**new** Color(0, 0, 0));  **this**.setLayout(**null**);  **this**.setHeading(heading);  **this**.setPosition(pos);  }  **public void** setHeading(String heading) {  **this**.heading = **new** JLabel(heading);  **this**.heading.setBounds(20, 0, 360, 70);  **this**.heading.setForeground(Color.white);  **this**.heading.setFont(**new** Font(**"Helvetica"**, Font.PLAIN, 36));  **this**.add(**this**.heading);  }  **public void** setPosition(**int** pos) {  **this**.position = pos;  **switch** (pos) {  **case** 0:  **this**.setBounds(50, 50, 400, 400);  **break**;  **case** 1:  **this**.setBounds(480, 50, 400, 400);  **break**;  **case** 2:  **this**.setBounds(50, 1120, 400, 400);  **break**;  **case** 3:  **this**.setBounds(480, 1120, 400, 400);  **break**;  **default**:  **break**;  }  }  **public** JLabel getHeading() { **return** heading; }  **public int** getPosition() { **return this**.position; }  **public** Widget getInstance() { **return this**; }  **public static** ImageIcon getImageIcon(String path, String description) {  java.net.URL imgURL = Widget.**class**.getResource(path);  **if** (imgURL != **null**) {  **return new** ImageIcon(imgURL, description);  } **else** {  System.err.println(**"Couldn't find file: "** + path);  **return null**;  }  } } |

Codeabschnitt 114: Überklasse der Widgets (*Widget.java*)

Im Custom-Konstruktor *Widget(...)* werden die Funktionen *setHeading(...)* und *setPosition(...)* aufgerufen.

Die Methode *setHeading(...)* kann zum Darstellen einer Überschrift verwendet werden. Überschriften werden allerdings nicht benötigt und deshalb ist die Funktion eigentlich überflüssig.

Die Funktion *setPosition(...)*, definiert je nach dem Inhalt der Variable pos, die richtige Anordnung des Widgets in Pixel.

|  |  |  |
| --- | --- | --- |
| **Position** | **Widget-Anordnung** | **Anordnung** (x, y, Länge x Höhe) |
| 0 | links oben | 50, 50, 400 x 400 |
| 1 | rechts oben | 480, 50, 400 x 400 |
| 2 | links unten | 50, 1120, 400 x 400 |
| 3 | rechts unten | 480, 1120, 400 x 400 |

* + 1. ReflectonUI/com/pinnovations/widgets/EmailWidget.java

Das E-Mail-Widget führt in Fettschrift den gewissen Absender an. Darunter findet sich der zugehörige Betreff in Normalschrift.

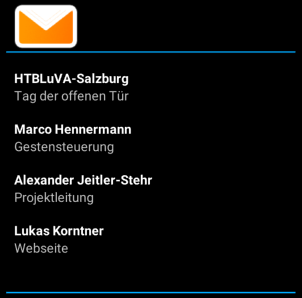


Abbildung 44: E-Mail-Widget

Die Klasse *EmailWidget* erbt von der Klasse *Widget*. Im unten abgebildeten Codeblock lässt sich dies auch erkennen.

|  |
| --- |
| **public class** EmailWidget **extends** Widget {  **private static final long** serialVersionUID = 1L;  **private** JTable table;  **private** JLabel icon;  **private** Email[] emails;    **public** EmailWidget() { **super**(**""**, -1); }  **public** EmailWidget(**int** pos, Email[] emails) {  **super**(**""**, pos);  **this**.emails = emails;  **this**.setBackgroundPicture();  **this**.setEmailTable();  }  **public** EmailWidget(String heading, **int** pos, Email[] emails) {  **super**(heading, pos);  **this**.emails = emails;  **this**.setBackgroundPicture();  **this**.setEmailTable();  }  **public void** setBackgroundPicture() {  ImageIcon image = Widget.getImageIcon(ImageConfig.pathEmailWidget, **""**);  **this**.icon = **new** JLabel();  **this**.icon.setBounds(0, 0, 400, 400);  **this**.icon.setIcon(image);  **this**.add(**this**.icon);  }  **public void** setEmailTable() {  **if**(**this**.emails.length > 0) {  **this**.table = **new** JTable();  **this**.table.setBounds(10, 80, 380, 300);  **this**.table.setBackground(Color.BLACK);  **this**.table.setForeground(Color.LIGHT\_GRAY);  **this**.table.setGridColor(Color.BLACK);  **this**.table.setRowHeight(70);  Font font = FontSupport.getFont(FontConfig.pathRobotoRegular, 18f);  DefaultTableCellRenderer cellRenderer = TableSupport  .getTableCellRenderer(font);  **for** (**int** i = 0; i < **this**.emails.length; i++) {  **if**(**this**.emails[i].getFrom().contains(**" <"**)) {  **int** index = **this**.emails[i].getFrom().indexOf(**" <"**);  **if**(index != -1) {  **this**.emails[i].setFrom(**this**.emails[i].getFrom()  .substring(0, index));  }  }  }  Object[][] tableRowData = **new** Object[**this**.emails.length][1];  **for** (**int** i = 0; i < tableRowData.length; i++) {  tableRowData[i][0] = TableSupport.getFormattedTextWithHeading(  **"<font size='5.5' color='white'><b>"** + **this**.emails[i].getFrom()  + **"</b></font>"**, **this**.emails[i].getSubject(), font, 360);  }  **this**.table.setModel(**new** DefaultTableModel(  tableRowData,  **new** String[] {  **"Subject"** }  ));  **this**.table.getColumnModel().getColumn(0).setCellRenderer(cellRenderer);  **this**.icon.add(**this**.table);  }  } } |

Codeabschnitt 115: Klasse des E-Mail-Widgets (*EmailWidget.java*)

Innerhalb des Konstruktors werden die Funktionen *setBackgroundPicture()*, sowie *setEmailTable()* aufgerufen. Die *setBackgroundPicture()*-Funktion ist für den Hintergrund, also für das E-Mail-Symbol und die blauen Trennlinien verantwortlich. Hingegen definiert die Funktion *setEmailTable()* den Inhalt der Tabelle.

* + 1. ReflectonUI/com/pinnovations/widgets/CalendarWidget.java

Das Kalender-Widget zeigt in der linken Spalte der Tabelle die Benennung des Ereignisses an. In der rechten Spalte hingegen werden das Datum, die Uhrzeit und der Ort dargestellt.

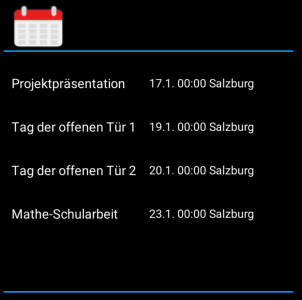


Abbildung 45: Kalender-Widget

Genauso wie beim Email-Widget, besteht auch das Kalender-Widget aus einem Hintergrund und einer Tabelle. Dies wird mit einem Blick auf die unten abgebildeten Klasse verdeutlicht.

|  |
| --- |
| **public class** CalendarWidget **extends** Widget {  **private static final long** serialVersionUID = 1L;  **private** JTable table;  **private** JLabel icon;  **private** Appointment[] appointments;    **public** CalendarWidget() {  **super**(**""**, -1);  }  **public** CalendarWidget(**int** pos, Appointment[] appointments) {  **super**(**""**, pos);  **this**.appointments = appointments;  **this**.setBackgroundPicture();  **this**.setAppointmentTable();  }  **public** CalendarWidget(String heading, **int** pos, Appointment[] appointments) {  **super**(heading, pos);  **this**.appointments = appointments;  **this**.setBackgroundPicture();  **this**.setAppointmentTable();  }    **public void** setBackgroundPicture() {  ImageIcon image = Widget.getImageIcon(ImageConfig.pathCalendarWidget, **""**);  **this**.icon = **new** JLabel();  **this**.icon.setBounds(0, 0, 400, 400);  **this**.icon.setIcon(image);  **this**.add(**this**.icon);  }  **public void** setAppointmentTable() {  **if**(**this**.appointments.length > 0) {  **this**.table = **new** JTable();  **this**.table.setBounds(10, 80, 380, 300);  **this**.table.setBackground(Color.BLACK);  **this**.table.setForeground(Color.WHITE);  **this**.table.setGridColor(Color.BLACK);  **this**.table.setRowHeight(60);  Font titleFont = FontSupport.getFont(FontConfig.pathRobotoRegular, 18f);;  DefaultTableCellRenderer titleCellRenderer = TableSupport  .getTableCellRenderer(titleFont);  Font dateAndLocationFont = FontSupport.getFont(FontConfig.pathRobotoRegular,   16f);  DefaultTableCellRenderer dateAndLocationCellRenderer = TableSupport  .getTableCellRenderer(dateAndLocationFont);  Object[][] tableRowData = **new** Object[**this**.appointments.length][2];  **for** (**int** i = 0; i < tableRowData.length; i++) {  tableRowData[i][0] = TableSupport.getFormattedText(  **this**.appointments[i].getTitle(), titleFont, 180);  String date = **this**.appointments[i].getStartDate();  String[] dateArray = date.split(**","**);  String day = **""**;  String month = **""**;  String hour = **""**;  String minute = **""**;  **for** (**int** j = 0; j < dateArray.length; j++) {  dateArray[j] = dateArray[j].replaceAll(**" "**, **""**);  **if**(j == 2) {  month = dateArray[j];  }  **else if**(j == 3) {  day = dateArray[j];  }  **else if**(j == 4) {  **if** (dateArray[j].length() < 2) {  dateArray[j] = **"0"** + dateArray[j];  }  hour = dateArray[j];  }  **else if**(j == 5) {  **if** (dateArray[j].length() < 2) {  dateArray[j] = **"0"** + dateArray[j];  }  minute = dateArray[j];  }  }  date = day + **"."** + month + **"."** + **" "** + hour + **":"** + minute;  String location = **this**.appointments[i].getLocation();  **if**(location.equals(**"None"**) || location.equals(**"none"**)) {  location = **""**;  }  String dateAndLocation = date + **" "** + location;  tableRowData[i][1] = TableSupport.getFormattedText(dateAndLocation,   dateAndLocationFont, 180);  }  **this**.table.setModel(**new** DefaultTableModel(  tableRowData,  **new** String[] {  **"Title"**, **"DateAndLocation"** }  ));  **this**.table.getColumnModel().getColumn(0).setCellRenderer(titleCellRenderer);  **this**.table.getColumnModel().getColumn(1)  .setCellRenderer(dateAndLocationCellRenderer);  **this**.icon.add(**this**.table);  }  } } |

Codeabschnitt 116: Klasse des Kalender-Widgets (*CalendarWidget.java*)

Die Funktionen *setBackgroundPicture()* und *setAppointmentTable()* der Kalender-Widget-Klasse sind denen in der E-Mail-Widget-Klasse sehr ähnlich. Die *setBackgroundPicture()*-Funktion ist wieder für den Hintergrund, also für das Kalender-Symbol und die blauen Trennlinien verantwortlich. Hingegen ist die *setAppointmentTable()* etwas komplexer, da zwei Spalten auftreten.

* + 1. ReflectonUI/com/pinnovations/widgets/WeatherWidget.java

Das Wetter-Widget dient zur Anzeige der aktuellen Temperatur, sowie der aktuellen Wetterlage.



Abbildung 46: Wetter-Widget

Das, was den unten abgebildeten Code so lang macht ist, dass die Position vom Temperaturtext, sowie vom Bild an eine bestimmte Widget-Anordnung angepasst werden muss.

|  |
| --- |
| **public class** WeatherWidget **extends** Widget {  **private static final long** serialVersionUID = 1L;  **private** JLabel temperature, icon;  **private** Day currentWeather;    **public** WeatherWidget() {  **super**(**""**, -1);  }    **public** WeatherWidget(**int** pos, Day currentWeather) {  **super**(**""**, pos);  **this**.currentWeather = currentWeather;  **this**.initializeWeatherWidget();  **this**.setWeatherWidget();  }    **public** WeatherWidget(String heading, **int** pos, Day currentWeather) {  **super**(heading, pos);  **this**.currentWeather = currentWeather;  **this**.initializeWeatherWidget();  **this**.setWeatherWidget();  }    **public void** setWeather(Day currentWeather) {  **this**.currentWeather = currentWeather;  }   **public void** initializeWeatherWidget() {  **this**.temperature = **new** JLabel();  **this**.temperature.setForeground(Color.white);  **this**.temperature.setFont(FontSupport.getFont(FontConfig.pathRobotoRegular,   100f));  **int** temperatureX, temperatureY;  temperatureX = 150;  temperatureY = 30;    **switch** (**this**.getPosition()) {  **case** 0:  **case** 2:  **this**.temperature.setBounds(0, temperatureY, **this**.getWidth(), 100);  **break**;  **case** 1:  **case** 3:  **this**.temperature.setBounds(300, temperatureY, **this**.getWidth(), 100);  **break**;    **default**:  **break**;  }  **this**.add(**this**.temperature);  **this**.icon = **new** JLabel();  **this**.add(**this**.icon);  }    **public void** setWeatherWidget() {  String[] fields = **this**.currentWeather.getFields();  **int** posTemperature = -1;  **for** (**int** i = 0; i < fields.length; i++) {  **if**(fields[i].contains(**"temperature"**)) {  posTemperature = i;  }  }  String fieldTemperature = fields[posTemperature].replace(**"temperature: "**, **""**);  fieldTemperature = fieldTemperature.replace(**"\""**, **""**);  String[] temperatureArray = fieldTemperature.split(**"\\."**);  **int** temperatureInteger = Integer.parseInt(temperatureArray[0]);  **int** temperaturePointNumbers = Integer.parseInt(temperatureArray[1]);    **if**(temperaturePointNumbers >= 50) {  temperatureInteger++;  }  **this**.temperature.setText(Integer.toString(temperatureInteger) + **"°"**);    String imageName = **""**;  **int** posIcon = -1;  **for** (**int** i = 0; i < fields.length; i++) {  **if**(fields[i].contains(**"icon"**)) {  posIcon = i;  }  }  String field = fields[posIcon].replace(**"icon: "**, **""**);  field = field.replace(**"\""**, **""**);    **if**(field.equals(**"clear-day"**)) {  imageName = ImageConfig.pathWeatherWidgetClear;  }  **else if**(field.equals(**"wind"**)) {  imageName = ImageConfig.pathWeatherWidgetWind;  } **...**  ImageIcon image = Widget.getImageIcon(imageName, **""**);    **int** iconX, iconY;  iconX = 40;  iconY = 90;  **switch** (**this**.getPosition()) {  **case** 0:  **case** 2:  **this**.icon.setBounds(iconX, iconY, image.getIconWidth(),  image.getIconHeight());  **break**;  **case** 1:  **case** 3:  **this**.icon.setBounds(120, iconY, image.getIconWidth(),   image.getIconHeight());  **break**;    **default**:  **break**;  }  **this**.icon.setIcon(image);  } } |

Codeabschnitt 117: Klasse des Wetter-Widgets (*WeatherWidget.java*)

Die Funktion *initializeWeatherWidget()* ist zur Initialisierung des Temperaturtexts und des Bildes verantwortlich. Die eigentliche Darstellungsart wird mithilfe der Funktion *setWeatherWidget()* festgelegt.

* + 1. ReflectonUI/com/pinnovations/widgets/ClockWidget.java

Das wohl unkomplizierteste Widget stellt das Uhren-Widget dar. Es besteht lediglich aus Datum und Uhrzeit.



Abbildung 47: Uhren-Widget

Auch bei diesem Widget muss sich die Anordnung innerhalb des Widgets bei veränderter Widget-Anordnung ändern. Dies ist auch im unten abgebildeten Codeblock recht gut erkennbar.

|  |
| --- |
| **public class** ClockWidget **extends** Widget {  **private static final long** serialVersionUID = 1L;  **private** JLabel date, time;  **private** Clock clock;    **public** ClockWidget() {  **super**(**""**, -1);  **this**.initializeClockWidget();  }  **public** ClockWidget(**int** pos, Clock clock) {  **super**(**""**, pos);  **this**.clock = clock;  **this**.initializeClockWidget();  **this**.setClockWidget();  }  **public** ClockWidget(String heading, **int** pos, Clock clock) {  **super**(heading, pos);  **this**.clock = clock;  **this**.initializeClockWidget();  **this**.setClockWidget();  }  **public void** setClock(Clock clock) {  **this**.clock = clock;  }  **public void** initializeClockWidget() {  **this**.date = **new** JLabel();  **this**.date.setForeground(Color.white);  **this**.date.setFont(FontSupport.getFont(FontConfig.pathRobotoRegular, 28f));  **this**.time = **new** JLabel();  **this**.time.setForeground(Color.white);  **this**.time.setFont(FontSupport.getFont(FontConfig.pathRobotoThin, 72f));  **int** dateX, dateY, timeX, timeY;  **switch** (**this**.getPosition()) {  **case** 0:  **case** 2:  dateX = 15;  dateY = 10;  **this**.date.setBounds(dateX, dateY, **this**.getWidth()-dateX, 50);  timeX = 10;  timeY = 60;  **this**.time.setBounds(timeX, timeY, **this**.getWidth()-timeX, 100);  **break**;  **case** 1:  **case** 3:  dateX = 160;  dateY = 20;  **this**.date.setBounds(dateX, dateY, **this**.getWidth()-dateX, 50);  timeX = 180;  timeY = 60;  **this**.time.setBounds(timeX, timeY, **this**.getWidth()-timeX, 100);  **break**;    **default**:  **break**;  }  **this**.add(**this**.date);  **this**.add(**this**.time);  }  **public void** setClockWidget() {  String[] dateArray = **this**.clock.getDate().split(**"-"**);  String day = **""**;  String suffix = **""**;  String month = **""**;    **if**(dateArray.length > 1) {  day = dateArray[0];  **if**(dateArray[0].charAt(0) == **'0'**) {  day = day.substring(1);  }  **switch** (dateArray[0]) {  **case "01"**:  **case "21"**:  **case "31"**:  suffix = **"st"**;  **break**;  ...  **default**:  suffix = **"th"**;  **break**;  }  **switch** (dateArray[1]) {  **case "01"**:  month = **"January"**;  **break**;  ...  **default**:  **break**;  }  }  **this**.date.setText(**"<html>"** + day + **"<font size=3> <sup>"** + suffix  + **"</sup></font> of "** + month + **"</html>"**);  **this**.time.setText(clock.getTime());  } } |

Codeabschnitt 118: Klasse des Uhren-Widgets (*ClockWidget.java*)

Ähnlich wie beim Wetter-Widget ist die Funktion *initializeClockWidget()* zur Initialisierung von beiden Texten (Datum und Uhrzeit) verantwortlich. Die eigentliche Darstellungsart wird mithilfe der Funktion *setClockWidget()* festgelegt.

* 1. **Ansichten**

Die Java-Benutzeroberfläche ist aus verschiedenen *Views* (engl. Ansichten) aufgebaut. Diese *Views* leiten von der Klasse *JPanel* ab. Die Ansichten werden auf einem Frame (in diesem Fall *MainFrame*) dargestellt und können gewechselt werden.

* + 1. ReflectonUI/com/pinnovations/views/MainView.java

Die Hauptansicht besteht aus den Widgets Wetter, Uhr, Email und Kalender, welche alle in einer der vier Ecken angeordnet sind. Die Anordnung ist in der Datenbank gespeichert und durch die iOS-App somit steuerbar.

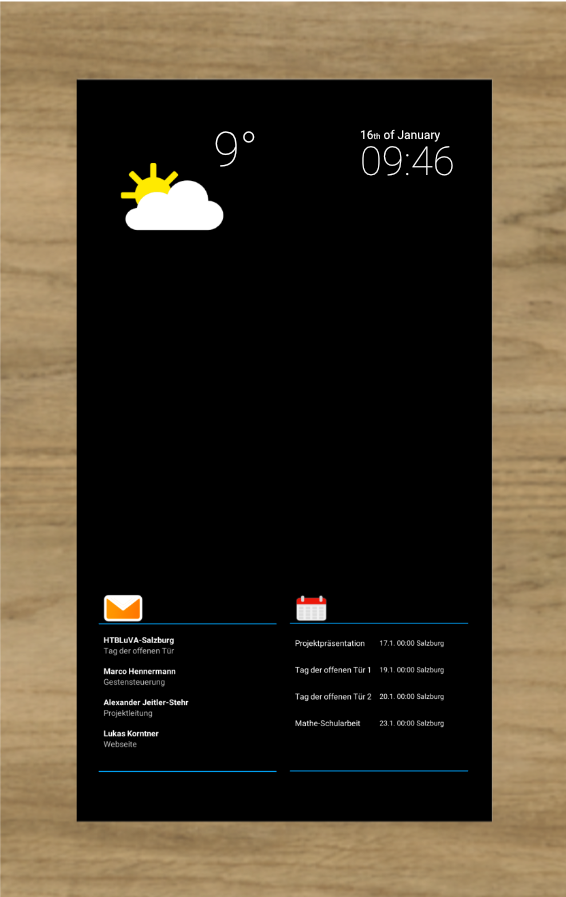


Abbildung 48: Hauptansicht

Zu Anfang erklären wir die oben abgebildete Hauptansicht *MainView*, welche von der Klasse *JPanel* erbt. Diese Klasse wird aufgrund ihrer Länge in mehreren Codeabschnitten unten abgebildet. Eine Instanz der Klasse *MainView* kann schlussendlich einem Objekt des Typs *JFrame* mit *add()* hinzugefügt werden.

Membervariablen

Die Membervariablen sind vom jeweiligen Typ der einzelnen Widgets. Außerdem gibt es noch eine Hilfsvariable vom Typ String.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| emailWidget | Das Email-Widget wird mit einem *Email*-Array befüllt, wodurch es die letzten Emails darstellen kann. |
| calendarWidget | Das Kalender-Widget wird mit einem *Appointment*-Array befüllt, wodurch die nächsten aufkommenden Termine abgebildet werden können. |
| weatherWidget | Dieses Widget dient zur Darstellung des aktuellen Wetters. Dafür sind Daten notwendig, welche mithilfe eines *Day*-Objekts übergeben werden. |
| clockWidget | Genauso wie bei den vorhergehenden Membervariablen, müssen Daten übergeben werden. Dies geschieht beim Uhren-Widget mithilfe eines *Clock*-Objekts. |
| widgetsArrangement- DividedBySlashes | Diese Variable ist lediglich zur Speicherung der Widget-Anordnung notwendig. |

Funktionen

Unter den Funktionen finden sich der Konstruktor *MainView()*, sowie die Methoden *drawWidgets()* und *updateView()*.

Der Konstruktor *MainView()* ist für die Initialisierung der Membervariablen der Superklasse *JPanel* über Setter-Funktionen verantwortlich. Außerdem wird in seinem Block auch noch die Hilfsvariable wid *widgetsArrangementDividedBySlashes* initialisiert.

|  |
| --- |
| **public** MainView() {  **this**.setSize(1080, 1920);  **this**.setLayout(**null**);  **this**.setBackground(Color.BLACK);  String[] widgetsArrangementArray = InterfaceDownload.getWidgetsArrangement(  FileConfig.reflectonEmailPath);  **this**.widgetsArrangementDividedBySlashes = **""**;  **for** (**int** i = 0; i < widgetsArrangementArray.length; i++) {  **this**.widgetsArrangementDividedBySlashes += widgetsArrangementArray[i];  **if**(i != widgetsArrangementArray.length-1) {  **this**.widgetsArrangementDividedBySlashes += **"/"**;  }  }  **this**.drawWidgets();  **this**.updateView(); } |

Codeabschnitt 119: Konstruktor der Hauptansicht (*MainView.java*)

Im ersten Abschnitt des Codes werden *JPanel*-spezifische Parameter wie beispielsweise Größe gesetzt. Danach folgt das Herunterladen der Widget-Anordnung mithilfe der *InterfaceDownload*-Klasse. Die Anordnung wird folglich formatiert und in der Membervariable *widgetsArrangementDividedBySlashes* gespeichert. Zum Schluss werden noch die Funktionen *drawWidgets()* und *updateView()* aufgerufen, sodass beim Instanziieren eines Objektes der *MainView*-Klasse die Widgets automatisch gezeichnet werden und die Ansicht automatisch aktualisiert wird.

Die Funktion *drawWidgets()* wird, wie der Name schon sagt, zum Zeichnen der verschiedenen Widgets verwendet. Abhängig von der Widget-Anordnung, welche in *widgetsArrangementDividedBySlashes* gespeichert ist, werden die Widgets entweder links oben, rechts oben, links unten, oder rechts unten ausgerichtet.

|  |
| --- |
| **public void** drawWidgets() {  String[] widgetsArrangementArray = InterfaceDownload.getWidgetsArrangement(  FileConfig.reflectonEmailPath);  **this**.removeAll();  **for** (**int** i = 0; i < widgetsArrangementArray.length; i++) {  **if** (widgetsArrangementArray[i].contains(**"email"**)) {  **if**(widgetsArrangementArray[i].contains(**":"**)) {  String[] emailPasswordArray = widgetsArrangementArray[i].split(**":"**);  String email = emailPasswordArray[emailPasswordArray.length-2];  String password = emailPasswordArray[emailPasswordArray.length-1];  Email[] emails = **null**;  **try** {  emails = InterfaceDownload.getEmails(email, password, 4);  } **catch** (MessagingException e) {  }  emailWidget = **new** EmailWidget(i, emails);  **this**.add(emailWidget);  }  }  **if** (widgetsArrangementArray[i].contains(**"calendar"**)) {  **if**(widgetsArrangementArray[i].contains(**":"**)) {  SimpleDateFormat sdf = **new** SimpleDateFormat(**"YYYY-MM-dd"**);  String currentDate = sdf.format(**new** Date());    String[] emailPasswordArray = widgetsArrangementArray[i].split(**":"**);  String email = emailPasswordArray[emailPasswordArray.length-2];  String password = emailPasswordArray[emailPasswordArray.length-1];  Appointment[] appointments = InterfaceDownload.getAppointments(  email, password, currentDate);calendarWidget = **new** CalendarWidget(i, appointments);  **this**.add(calendarWidget);  }  }  **if** (widgetsArrangementArray[i].contains(**"weather"**)) {  GeoLocation location = InterfaceDownload.getLocation();  Day currentWeather = InterfaceDownload.getCurrentWeather(  location.getLatitude(),location.getLongitude());  weatherWidget = **new** WeatherWidget(i, currentWeather);  **this**.add(weatherWidget);  }  **if** (widgetsArrangementArray[i].contains(**"clock"**)) {  Clock dateAndTime = InterfaceDownload.getClock();  clockWidget = **new** ClockWidget(i, dateAndTime);  **this**.add(clockWidget);  }  }  **this**.revalidate();  **this**.repaint(); } |

Codeabschnitt 120: Funktion zum Zeichnen der Widgets (*MainView.java*)

Zuerst wird die Widget-Anordnung in ein Array gespeichert. Jenes Array wird mithilfe einer for-Schleife durchlaufen. Je nachdem bei welchem Schleifendurchlauf ein Element auftritt, so hat dies eine Auswirkung auf die Anordnung aller Widgets. Tritt das Email-Widget also beispielsweise bei *i=2* auf, so befindet sich das Widget links unten.

|  |  |
| --- | --- |
| **Zählvariable** i | **Widget-Anordnung** |
| 0 | links oben |
| 1 | rechts oben |
| 2 | links unten |
| 3 | rechts unten |

Innerhalb jeder der einzelnen vier if-Anweisungen lässt sich sehen, dass die Position i, also die Anordnung, an das bestimmte Widget im Konstruktor übergeben wird. Außerdem muss dem Konstruktor bei der Instanziierung des Widgets auch noch eine Variable übergeben werden, welche Widget-spezifische Daten enthält. Diese Daten werden meist aus der *InterfaceDownload*-Klasse gewonnen.

Um die Ansicht auf einem halbwegs aktuellen Stand zu halten wird die Funktion *updateView()* benötigt. Jene Funktion enthält mehrere Threads, wodurch bestimmte programmrelevante Bestandteile in einer spezifischen Periode aktualisiert werden.

|  |
| --- |
| **public void** updateView() {  Timer checkNewArrangement = **new** Timer();  checkNewArrangement.schedule(**new** TimerTask() {  **public void** run() {  String[] widgetsArrangementArrayNew = InterfaceDownload  .getWidgetsArrangement(FileConfig.reflectonEmailPath);  String widgetsArrangementDividedBySlashesNew = **""**;  **for** (**int** i = 0; i < widgetsArrangementArrayNew.length; i++) {  widgetsArrangementDividedBySlashesNew += widgetsArrangementArrayNew[i];  **if**(i != widgetsArrangementArrayNew.length-1) {  widgetsArrangementDividedBySlashesNew += **"/"**;  }  }  **if**(!widgetsArrangementDividedBySlashesNew.equals(  widgetsArrangementDividedBySlashes)) {  widgetsArrangementDividedBySlashes =   widgetsArrangementDividedBySlashesNew;  drawWidgets();  }  }  }, 0, 1000);Timer updateAfterTime = **new** Timer();  updateAfterTime.schedule(**new** TimerTask() {  **public void** run() {  drawWidgets();  }  }, 0, 600000);  Timer updateWeatherWidget = **new** Timer();  updateWeatherWidget.schedule(**new** TimerTask() {  **public void** run() {  GeoLocation location = InterfaceDownload.getLocation();  Day currentWeather = InterfaceDownload.getCurrentWeather(  location.getLatitude(),location.getLongitude());  weatherWidget.setWeather(currentWeather);  weatherWidget.setWeatherWidget();  }  }, 0, 60000);  Timer updateClockWidget = **new** Timer();  updateClockWidget.schedule(**new** TimerTask() {  **public void** run() {  Clock dateAndTime = InterfaceDownload.getClock();  clockWidget.setClock(dateAndTime);  clockWidget.setClockWidget();  }  }, 0, 1000); } |

Codeabschnitt 121: Funktion zum Aktualisieren der Ansicht (*MainView.java*)

Der Thread *checkNewArrangement* lädt jede Sekunde (1000ms) die Anordnung der Widgets herunter, welche durch die Smartphone-App umgestellt werden kann.   
Sollte die neu geladene Widget-Anordnung von jener in der Membervariable *widgetsArrangementDividedBySlashes* gespeicherten abweichen, so wird die Membervariable mit dem neuen Wert überschrieben.

Außerdem ist es äußerst hilfreich, eine Aktualisierung der kompletten Anzeige alle zehn Minuten durchzuführen. Für dies ist der Thread *updateAfterTime* verantwortlich.

Um reine Daten wie z.B. das Wetter herunterzuladen, werden die Threads *updateWeatherWidget*, sowie *updateClockWidget* benötigt.

* + 1. ReflectonUI/com/pinnovations/views/EmailView.java

Aus Zeitgründen zeigt die E-Mail-App bzw. E-Mail-Ansicht in der derzeitigen Version nicht mehr Informationen wie die Widgets an.



Abbildung 49: E-Mail-App

Außerdem ist das Design noch nicht optimiert, was bei einer möglichen Weiterführung des Projektes behoben werden kann.

|  |
| --- |
| **public class** EmailView **extends** JPanel {  **private** JTable table;  **private** Email[] emails;    **public** EmailView() {  **this**.setSize(1080, 1920);  **this**.setLayout(**null**);  **this**.setBackground(Color.BLACK);  **this**.drawTable();  }  **public void** drawTable() {  emails = **null**;  String[] widgetsArrangementArray = InterfaceDownload  .getWidgetsArrangement(FileConfig.reflectonEmailPath);  **this**.removeAll();  **for** (**int** i = 0; i < widgetsArrangementArray.length; i++) {  **if** (widgetsArrangementArray[i].contains(**"email"**)) {  **if**(widgetsArrangementArray[i].contains(**":"**)) {  String[] emailPasswordArray = widgetsArrangementArray[i].split(**":"**);  String email = emailPasswordArray[emailPasswordArray.length-2];  String password = emailPasswordArray[emailPasswordArray.length-1];  **try** {  emails = InterfaceDownload.getEmails(email, password, 5);  } **catch** (MessagingException e) { }  }  }  }  **if**(emails.length > 0) {  **this**.table = **new** JTable();  **this**.table.setBounds(40, 300, 1000, 600);  **this**.table.setBackground(Color.BLACK);  **this**.table.setForeground(Color.WHITE);  **this**.table.setGridColor(Color.BLACK);  **this**.table.setRowHeight(100);  **for** (**int** i = 0; i < emails.length; i++) {  **if**(emails[i].getFrom().contains(**" <"**)) {  **int** index = emails[i].getFrom().indexOf(**" <"**);  **if**(index != -1) {  emails[i].setFrom(emails[i].getFrom().substring(0, index));  }  }  }  Font fromFont = FontSupport.getFont(FontConfig.pathRobotoThin, 16f);  DefaultTableCellRenderer fromCellRenderer = TableSupport  .getTableCellRenderer(fromFont);  Font subjectFont = FontSupport.getFont(FontConfig.pathRobotoThin, 16f);  DefaultTableCellRenderer subjectCellRenderer = TableSupport  .getTableCellRenderer(subjectFont);  Font contentFont = FontSupport.getFont(FontConfig.pathRobotoThin, 16f);  DefaultTableCellRenderer contentCellRenderer = TableSupport  .getTableCellRenderer(contentFont);  Object[][] tableRowData = **new** Object[emails.length][3];  **for** (**int** i = 0; i < tableRowData.length; i++) {  tableRowData[i][0] = TableSupport.getFormattedText(  emails[i].getFrom(), fromFont, 200);  tableRowData[i][1] = TableSupport.getFormattedText(  emails[i].getSubject(),subjectFont, 300);  tableRowData[i][2] = TableSupport.getFormattedText(  emails[i].getContent(), contentFont, 500);  }  **this**.table.setModel(**new** DefaultTableModel(  tableRowData,  **new** String[] {  **"From"**, **"Subject"**, **"Content"** }  ));   **this**.table.getColumnModel().getColumn(0)  .setCellRenderer(fromCellRenderer);  **this**.table.getColumnModel().getColumn(1)  .setCellRenderer(subjectCellRenderer);  **this**.table.getColumnModel().getColumn(2)  .setCellRenderer(contentCellRenderer);  **this**.add(**this**.table);  }  } } |

Codeabschnitt 122: Klasse der E-Mail-Ansicht (*EmailView.java*)

Die Klasse besteht aus dem Konstruktor *EmailView()*, welcher bestimmte Membervariablen der Superklasse via Setter-Funktion setzt und die Funktion *drawTable()* aufruft. Diese Funktion erstellt nach dem Erhalt von Daten, in diesem Fall E-Mails, eine Instanz der Klasse *JTable*. Diese Tabelle wird dann schlussendlich mit etwaigen Werten befüllt.

* + 1. ReflectonUI/com/pinnovations/views/CalendarView.java

Die Kalender-App ist ebenfalls noch nicht fertiggestellt. Da sich das Look & Feel noch in einem Anfangsstadium befindet, wird auf ein Bild verzichtetet und nur der Code der aktuellen Version abgebildet.

|  |
| --- |
| **public class** CalendarView **extends** JPanel {  **private** JTable table;  **private** Appointment[] appointments;    **public** CalendarView() {  **this**.setSize(1080, 1920);  **this**.setLayout(**null**);  **this**.setBackground(Color.BLACK);  **this**.drawTable();  }    **public void** drawTable() {  appointments = **null**;  String[] widgetsArrangementArray = InterfaceDownload  .getWidgetsArrangement(FileConfig.reflectonEmailPath);  **this**.removeAll();  **for** (**int** i = 0; i < widgetsArrangementArray.length; i++) {  **if** (widgetsArrangementArray[i].contains(**"calendar"**)) {  **if**(widgetsArrangementArray[i].contains(**":"**)) {  SimpleDateFormat sdf = **new** SimpleDateFormat(**"YYYY-MM-dd"**);  String currentDate = sdf.format(**new** Date());  String[] emailPasswordArray = widgetsArrangementArray[i].split(**":"**);  String email = emailPasswordArray[emailPasswordArray.length-2];  String password = emailPasswordArray[emailPasswordArray.length-1];  appointments = InterfaceDownload.getAppointments(email, password,  currentDate);  }  }  }  **if**(**this**.appointments.length > 0) {  **this**.table = **new** JTable();  **this**.table.setBounds(10, 50, 380, 300);  **this**.table.setBackground(Color.BLACK);  **this**.table.setForeground(Color.WHITE);  **this**.table.setGridColor(Color.BLACK);  **this**.table.setRowHeight(60);    Font titleFont = **new** Font(**"Helvetica"**, Font.PLAIN, 20);  DefaultTableCellRenderer titleCellRenderer = TableSupport  .getTableCellRenderer(titleFont);    Font dateAndLocationFont = **new** Font(**"Helvetica"**, Font.PLAIN, 16);  DefaultTableCellRenderer dateAndLocationCellRenderer = TableSupport  .getTableCellRenderer(dateAndLocationFont);    Object[][] tableRowData = **new** Object[**this**.appointments.length][2];  **for** (**int** i = 0; i < tableRowData.length; i++) {  tableRowData[i][0] = TableSupport.getFormattedText(  **this**.appointments[i].getTitle(), titleFont, 180);  String date = **this**.appointments[i].getStartDate();  String[] dateArray = date.split(**","**);  String day = **""**;  String month = **""**;  String hour = **""**;  String minute = **""**;  **for** (**int** j = 0; j < dateArray.length; j++) {  dateArray[j] = dateArray[j].replaceAll(**" "**, **""**);  **if**(j == 2) {  month = dateArray[j];  }  **else if**(j == 3) {  day = dateArray[j];  }  **else if**(j == 4) {  **if** (dateArray[j].length() < 2) {  dateArray[j] = **"0"** + dateArray[j];  }  hour = dateArray[j];  }  **else if**(j == 5) {  **if** (dateArray[j].length() < 2) {  dateArray[j] = **"0"** + dateArray[j];  }  minute = dateArray[j];  }  }  date = day + **"."** + month + **"."** + **" "** + hour + **":"** + minute;  String location = **this**.appointments[i].getLocation();  **if**(location.equals(**"None"**) || location.equals(**"none"**)) {  location = **""**;  }  String dateAndLocation = date + **" "** + location;    tableRowData[i][1] = TableSupport.getFormattedText(dateAndLocation,   dateAndLocationFont, 180);  }  **this**.table.setModel(**new** DefaultTableModel(  tableRowData,  **new** String[] {  **"Title"**, **"DateAndLocation"** }  ));    **this**.table.getColumnModel().getColumn(0)  .setCellRenderer(titleCellRenderer);  **this**.table.getColumnModel().getColumn(1)  .setCellRenderer(dateAndLocationCellRenderer);  **this**.add(**this**.table);  }  } } |

Codeabschnitt 123: Klasse der Kalender-Ansicht (*CalendarView.java*)

Ebenfalls wie in der E-Mail-App wird in der Klasse *CalendarView* eine Tabelle abgebildet. Jene Tabelle ist vom Typ *JTable* und beinhaltet Kalendereinträge als Daten.

* + 1. ReflectonUI/com/pinnovations/views/WeatherView.java

Leider befindet sich die Wetter-App auch noch in einem anfänglichen Zustand. Bei einer Projektweiterführung könnte das Design weitgehend angepasst werden und weitere Daten wie z.B. Windgeschwindigkeit angezeigt werden. Die Daten sind zwar im Programm verfügbar, aber aus Zeitgründen scheiterte es an der passenden Darstellung, sodass in der App noch die gleichen Informationen wie im *MainView* angezeigt werden.

|  |
| --- |
| **public class** WeatherView **extends** JPanel {  **private** JLabel temperature, icon;  **private** Day currentWeather;    **public** WeatherView() {  **this**.setSize(1080, 1920);  **this**.setLayout(**null**);  **this**.setBackground(Color.BLACK);  **this**.draw();  }  **public void** draw() {  GeoLocation location = InterfaceDownload.getLocation();  currentWeather = InterfaceDownload.getCurrentWeather(location.getLatitude(),   location.getLongitude());  **this**.temperature = **new** JLabel();  **this**.temperature.setForeground(Color.white);  **this**.temperature.setFont(**new** Font(**"Helvetica"**, Font.PLAIN, 100));  **this**.temperature.setBounds(200, 30, **this**.getWidth()-200, 100);  **this**.temperature.setBounds(260, 30, **this**.getWidth()-260, 100);  **this**.add(**this**.temperature);  **this**.icon = **new** JLabel();  **this**.add(**this**.icon);    String[] fields = **this**.currentWeather.getFields();  **int** posTemperature = -1;  **for** (**int** i = 0; i < fields.length; i++) {  **if**(fields[i].contains(**"temperature"**)) {  posTemperature = i;  }  }  String fieldTemperature = fields[posTemperature].replace(**"temperature: "**, **""**);  fieldTemperature = fieldTemperature.replace(**"\""**, **""**);  String[] temperatureArray = fieldTemperature.split(**"\\."**);  **int** temperatureInteger = Integer.parseInt(temperatureArray[0]);  **int** temperaturePointNumbers = Integer.parseInt(temperatureArray[1]);  **if**(temperaturePointNumbers >= 50) {  temperatureInteger++;  }  **this**.temperature.setText(Integer.toString(temperatureInteger) + **"°"**);  String imageName = **""**;  **int** posIcon = -1;  **for** (**int** i = 0; i < fields.length; i++) {  **if**(fields[i].contains(**"icon"**)) {  posIcon = i;  }  }  String field = fields[posIcon].replace(**"icon: "**, **""**);  field = field.replace(**"\""**, **""**);    **if**(field.equals(**"clear-day"**)) {  imageName = ImageConfig.pathWeatherWidgetClear;  }  **else if**(field.equals(**"wind"**)) {  imageName = ImageConfig.pathWeatherWidgetWind;  }  ...  ImageIcon image = createImageIcon(imageName, **""**);  **this**.icon.setBounds(40, 90, image.getIconWidth(), image.getIconHeight());  **this**.icon.setIcon(image);  }  **private** ImageIcon createImageIcon(String path, String description) {  java.net.URL imgURL = getClass().getResource(path);  **if** (imgURL != **null**) {  **return new** ImageIcon(imgURL, description);  } **else** {  System.err.println(**"Couldn't find file: "** + path);  **return null**;  }  } } |

Codeabschnitt 124: Klasse der Wetter-Ansicht (*WeatherView.java*)

Wie im obigen Programmteil ersichtlich, wird in der Funktion *draw()* das Widget gezeichnet. Dieses besteht im jetzigen Entwicklungsstand aus einem passenden Bild und einer Temperatur. Mit einer Reihe von if-Abfragen wird das korrekte Bild nach der aktuellen Wetterlage ausgewählt. Bei der Temperatur ist es einfacher, da sie einfach als Text an ein *JLabel* hinausgeschrieben wird.

* + 1. ReflectonUI/com/pinnovations/views/ClockView.java

Die Uhren-App stellt die einfachste Ansicht dar. Als Daten sind nur die aktuelle Zeit sowie das Datum anzuzeigen.

|  |
| --- |
| **public class** ClockView **extends** JPanel {  **private** JLabel date, time;  **private** Clock clock;    **public** ClockView() {  **this**.setSize(1080, 1920);  **this**.setLayout(**null**);  **this**.setBackground(Color.BLACK);  **this**.draw();  }  **public void** draw() {  String[] widgetsArrangementArray = InterfaceDownload  .getWidgetsArrangement(FileConfig.reflectonEmailPath);  **this**.removeAll();  **for** (**int** i = 0; i < widgetsArrangementArray.length; i++) {  **if** (widgetsArrangementArray[i].contains(**"clock"**)) {  clock = InterfaceDownload.getClock();  }  }  **this**.date = **new** JLabel();  **this**.date.setForeground(Color.white);  **this**.date.setFont(**new** Font(**"Helvetica"**, Font.PLAIN, 28));  **this**.time = **new** JLabel();  **this**.time.setForeground(Color.white);  **this**.time.setFont(**new** Font(**"Helvetica"**, Font.PLAIN, 72));  **this**.date.setBounds(15, 10, **this**.getWidth()-15, 50);  **this**.time.setBounds(10, 60, **this**.getWidth()-10, 100);  **this**.add(**this**.date);  **this**.add(**this**.time);    String[] dateArray = **this**.clock.getDate().split(**"-"**);  String day = **""**;  String suffix = **""**;  String month = **""**;  **if**(dateArray.length > 1) {  day = dateArray[0];  **if**(dateArray[0].charAt(0) == **'0'**) {  day = day.substring(1);  }  **switch** (dateArray[0]) {  **case "01"**:  **case "21"**:  **case "31"**:  suffix = **"st"**;  **break**;  ...  **default**:  suffix = **"th"**;  **break**;  }  **switch** (dateArray[1]) {  **case "01"**:  month = **"January"**;  **break**;  ...  **default**:  **break**;  }  }    **this**.date.setText(**"<html>"** + day + **"<font size=3> <sup>"** + suffix  + **"</sup></font> of "** + month + **"</html>"**);  **this**.time.setText(clock.getTime());  } } |

Codeabschnitt 125: Klasse der Uhren-Ansicht (*ClockView.java*)

Wie beim Wetter-Widget wird auch hier die Funktion *draw()* benötigt. Sie ermöglicht eine ordnungsgemäße Darstellung von Uhrzeit und Datum.

* 1. **Hauptframe und Hauptklasse**

Der Start des Java-Programmes befindet sich in der *main*-Funktion in der *ReflectonUI.java*-Datei. Diese Funktion instanziiert ein Objekt der *MainFrame*-Klasse. Jenem Frame wird mit der Funktion *add(...)* eine Ansicht (z.B. *MainView*) hinzugefügt. Außerdem befindet sich in der Ansicht *MainView* noch eine weitere Unterkategorie: Die Widgets.

* + 1. ReflectonUI/com/pinnovations/frames/MainFrame.java

Die MainFrame-Klasse ist die einzige Klasse im Projekt, welche von der Klasse JFrame erbt.

|  |
| --- |
| **public class** MainFrame **extends** JFrame {  **private static final long** serialVersionUID = 1L;  **public** MainFrame() {  **this**.setSize(**new** Dimension(1080, 1920));  **this**.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  **this**.setBackground(Color.BLACK);  **this**.setAutoRequestFocus(**false**);  **this**.setResizable(**false**);  **this**.setLocationRelativeTo(**null**);  **this**.setUndecorated(**true**);  **this**.getContentPane().setBackground(Color.BLACK);  **this**.getContentPane().setLayout(**null**);  }  **public void** clearFrame() {  SwingUtilities.invokeLater(() -> {  **this**.getContentPane().removeAll();  **this**.invalidate();  **this**.validate();  **this**.repaint();  });  } } |

Codeabschnitt 126: Klasse des Hauptframes (*MainFrame.java*)

Im oben abgebildeten Block lässt sich erkennen, dass die Klasse keine Erweiterung von *JFrame* ist, sondern nur bestimmte Werte der Superklasse vorweg setzt. Außerdem gibt es noch die Funktion *clearFrame()*, die zum Löschen des Inhalts befähigt.

* + 1. ReflectonUI/com/pinnovations/main/ReflectonUI.java

Zu guter Letzt folgt die Datei *ReflectonUI.java*, welche den absoluten Startpunkt darstellt. Die Klasse ist mit ihrer *main*-Funktion der Dreh- und Angelpunkt unseres Projektes.

|  |
| --- |
| **public static void** main(String[] args) {widgetsArrangementArray = InterfaceDownload  .getWidgetsArrangement(FileConfig.reflectonEmailPath);  gestureNumber = -1;  mainFrame = **new** MainFrame();  mainFrame.setVisible(**false**);  mainView = **new** MainView();  emailView = **new** EmailView();  calendarView = **new** CalendarView();  weatherView = **new** WeatherView();  clockView = **new** ClockView();  mainFrame.add(mainView);  mainFrame.setVisible(**true**);  Runnable gestureRunnable = **new** Runnable() {  @Override  **public void** run() {  **try** {  **final** Serial serial = SerialFactory.createInstance();  **try** {  SerialConfig config = **new** SerialConfig();  config.device(**"/dev/ttyACM0"**).baud(Baud.\_9600);  System.out.println(**"Connecting to: "** + config.toString());  serial.open(config);  }  **catch**(IOException ex) {  System.out.println(**" ==>> SERIAL SETUP FAILED : "** + ex.getMessage());  **return**;  }  serial.addListener(**new** SerialDataEventListener() {  **public void** dataReceived(SerialDataEvent event) {  **try** {  String text = event.getAsciiString();  **char** firstCharacter = text.charAt(0);  **if**(firstCharacter == **'-'** || firstCharacter == **'0'** || firstCharacter == **'1'** || firstCharacter == **'2'** ||   firstCharacter == **'3'**) {  **int** gestureNumberNew;  **switch**(firstCharacter){  **case '-'**:  gestureNumberNew = -1;  **break**;  **default**:  gestureNumberNew = Character.getNumericValue( firstCharacter);  **break**;  }  System.out.println(**"in int: "** + gestureNumberNew);  **if**(gestureNumberNew != gestureNumber) {  gestureNumber = gestureNumberNew;  mainFrame.getContentPane().removeAll();  **if**(gestureNumber >= 0) {  **if**(widgetsArrangementArray[gestureNumber] .contains(**"email"**)) {  mainFrame.add(emailView);  }  ...  **else if**(gestureNumber == -1) {  mainFrame.add(mainView);  }  mainFrame.revalidate();  mainFrame.repaint();  }  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  });  **for** (;;) {  Thread.sleep(500);  }  } **catch** (Exception e) {  System.out.println(e.getMessage());  }  }  };  Thread gestureThread = **new** Thread(gestureRunnable);  gestureThread.start();  Timer updateArrangement = **new** Timer();  updateArrangement.schedule(**new** TimerTask() {  **public void** run() {  widgetsArrangementArray = InterfaceDownload.getWidgetsArrangement( FileConfig.reflectonEmailPath);  }  }, 0, 1000); } |

Codeabschnitt 127: Hauptfunktion (*ReflectonUI.java*)

Nachdem diverse Instanziierungen bzw. Voreinstellungen getätigt wurden, kommt nun die Verknüpfung zwischen der Gestensteuerung, also dem Arduino, und dem Raspberry Pi ins Spiel.

Das Arduino schickt je nach ausgeführter Geste eine Zahl zwischen -1 und 3 über den seriellen Bus. Zahlen zwischen 0 und 3 werden gesendet, wenn vom Homescreen (MainView) in eine bestimmte App gesprungen wird. Die Zahl -1 wird gesendet, wenn aus der jeweiligen App zum Homescreen zurückgekehrt werden soll. Um den seriellen Strom der jeweiligen Zahlen auf dem Raspberry Pi zu empfangen, wird die Pi4J-Library benötigt. Mithilfe eines Threads gestureThread bzw. eigentlich der Funktion *serial.addListener(...)* werden die eben besprochenen Zahlen abgefragt. Die aufgerufene Ansicht (= View) hängt vom Wert der Zahl ab. Dies verdeutlichen auch die if-Anweisungen. Anders ausgedrückt bedeutet dies, dass je nach Richtung der Geste die Ansicht gewechselt wird.

Am Ende des Programmes findet sich noch ein Thread namens *updateArrangement* wieder. Jener ist dafür verantwortlich, die aktuelle Anordnung von der Datenbank aus in der Variable *widgetsArrangementArray* zu speichern.

1. **Gestensteuerung**

Um eine benutzerfreundliche Bedienung des Spiegels zu gewährleisten, wurde viel Zeit darin investiert, Überlegungen anzustellen, mit welcher Methodik die Bedienung am besten umzusetzen sein könnte. Dabei kamen einige verwirklichbare Ideen auf.

Zuerst wurde an einen Touchscreen gedacht. Dies hätte den Vorteil der präzisen und zuverlässigen Steuerbarkeit. Dennoch stand dieser Gedankengang nach geraumer Zeit vor dem Verwurf, da ein Touchscreen einen dauerhaft verschmierten Spiegel zur Folge hätte, wodurch sowohl die Sicht als auch die Ästhetik maßgeblich eingeschränkt würden.

Daraufhin wurde überlegt, ob es Sinn mache, den Touchscreen nach außerhalb des Spiegels zu verlegen. Dies würde bedeuten, dass ein kleines, etwa 12“ großes Touch-Display an der Seite des Spiegelrahmens angebracht werden müsste, in welchem dann alle Bedienmöglichkeiten vonstattengehen sollten. Da jedoch im Konsumentenbereich kaum jemand daran interessiert wäre, sich ein zusätzliches Gerät neben dem Spiegel zu hängen, um das Gerät steuern zu können, und dies außerdem zusätzliche Kosten verursachen würde, stieß auch diese Variante schlussendlich auf Ablehnung.

Eine weitere Variante der Bedienbarkeit wäre eine Sprachsteuerung gewesen. Diese wird in vielen derartigen Anwendungen verbaut und bringt diverse Vorteile mit sich. Unter anderem wären keinerlei Bewegungen nötig, wodurch der Wohlfühlfaktor vor dem Spiegel erhöht werden würde. Auf der anderen Seite ist die Spracherkennung problematisch, wenn man aus dialektsprachigen Regionen stammt. In Regionen wie dem Salzburger Land stellt dies eine große Hürde dar, da die meisten nicht unbedingt gewillt sind, in standard-deutsch Befehle anzusagen, da dies nicht ihre Alltagssprache ist. Aus diesem Grund wurde nach besseren Alternativen weitergeforscht, bevor auf diese Möglichkeit zurückgegriffen würde.

Dabei kam die Idee auf, eine Gestensteuerung zu entwickeln. Hierbei wurde geplant, mittels Wischgeste in Richtung der Anwendung zugehöriger Ecke den Befehl des Öffnens der entsprechenden Detailansicht auszuführen. Mit einem Wischen nach links unten wird dann die detaillierte Ansicht wieder geschlossen und das Hauptmenü geöffnet.

Die Bedienung per Hand bietet viele Vorteile und nur wenige Nachteile, sofern man den richtigen Sensor findet und diesen auf eine gelungene Art und Weise programmiert.

* 1. **Vor- und Nachteile**

Für die Verwirklichung einer Gestensteuerung spricht unter anderem die Einfachheit des Systems. Dadurch, dass nur vier Gesten interpretiert werden müssen, welche voneinander komplett unterschiedlich sind und allesamt für jegliche Art von Aufnahme sehr klar erkennbar sind, hat der Benutzer eine gewisse Freiheit beim Ausführen der Geste. Das heißt, er oder sie muss sich nicht bemühen, von einem Punkt zu einem anderen genau definierten Punkt zu wischen, sondern muss sich nur innerhalb eines 90°-Bereichs bewegen, um die Ansicht zu erreichen, die auch beabsichtigt wurde.

Gegenüber der Sprachsteuerung hat diese Variante den Vorteil, dass Gesten überall auf der Welt gleich aussehen und daher keine regionalen Adaptierungen durchgeführt werden müssen.

Weiters kommt hinzu, dass im Gegensatz zum Touch-Display hier kein zusätzliches Gerät bereitgestellt werden muss. Ganz im Gegenteil, es kommt nichts außer einem Sensor hinzu, welcher kaum größer als ein Fingernagel ist.

Dennoch bringt eine Gestensteuerung auch Nachteile mit sich. Beispielsweise sind gute Lichtverhältnisse von Nöten, damit die Gesten einwandfrei interpretiert werden können. Das bedeutet, keine zu starken Lichtquellen dürfen vom Hintergrund aus den Sensor anleuchten und es allgemein nicht zu dunkel sein darf.

Außerdem kann ein Problem dadurch entstehen, dass die geforderten Gesten nicht nur dann erkannt werden, wenn diese von einer menschlichen Hand stammen, sondern auch von anderen Gegenständen. Diese Problematik wird zwar bei einigen Sensoren technisch gelöst, aber nicht im kostengünstigen Sektor.

* 1. **Potenzielle Varianten und deren Nachteile**
     1. MGC3x30 GestIC

Eines der ersten Modelle zur Realisierung der Gestensteuerung, welches Potenzial zur Verwirklichung zeigte, war jenes von Digi-Key Electronics.[[11]](#footnote-12) Dies findet vor allem in der Unterhaltungselektronik Verwendung, etwa als kostengünstige Alternative zu kameragestützten Anwendungen mit komplexer Software.

Bei diesem System wird über mit Wechselspannung betriebene Elektroden ein elektrisches Feld über der Oberfläche eines Objekts erzeugt. Bewegungen in einer von der Fläche abhängigen Maximalhöhe über den Elektroden können dann präzise aufgenommen und interpretiert werden. Die maximale Erkennungshöhe würde im Fall eines 30x50cm großen Spiegels unter der Bedingung, dass dieser mit dem Feldwellensensor überdeckt wird, bei 12-20cm liegen, abhängig vom Sensortyp.

Die Feldlinien, die mit einem über dem Boden geerdeten Körper Kontakt aufnehmen, werden auf Masse gezogen und beeinflussen so das Gesamtfeld. Dies vermindert den Signalpegel, was eine Meldung beim Mikrocontroller zur Folge hat.

Da die Hardware das elektrische Massezentrum der menschlichen Hand erkennt, können Bewegungen verfolgt und so diverse Gesten voneinander unterschieden werden.

Der MGC3x30 GestIC stellt zwar allgemein eine kostengünstige und praktische Alternative zu kameragebundenen Systemen dar, spielt für die Verwirklichung dennoch keine Rolle, da die Reichweite mit maximal 20cm für unsere Bedürfnisse nicht zufriedenstellend ist.

Außerdem ist der Preis für Massenaufträge zwar relativ günstig, für ein Diplomarbeitsprojekt jedoch ungeeignet, da das komplette Entwicklungskit mit Mikrochip, Sensoren, Steckbrett, etc. rund 125€ kostet.

* + 1. Leap Motion

Der Preis war mit auch bei Leap Motion ein klarer Nachteil. Das Produkt bietet zwar einiges, aber für den konkreten Fall einer Gestensteuerung auf einem Smart-Mirror nicht mehr, als andere Optionen, welche kostengünstiger sind.[[12]](#footnote-13)

Die Hauptintention von Leap Motion ist es, beide Hände per Kamera, Sensoren und einem zusätzlichen VR-Gerät möglichst präzise aufzunehmen und in eine virtuelle Welt zu übertragen. In dieser virtuellen Welt kann man mit virtuellen Körpern Dinge konstruieren, die in der Realität nur schwer eigenhändig herzustellen wären.

Zusammenfassend kann man also sagen, dass Leap Motion zwar ein sehr nützliches Instrument darstellt, jedoch nicht für unsere Zwecke.

* + 1. Skywriter HAT

Dieser Sensor sah auf den ersten Blick ziemlich vielversprechend aus, da er Raspberry Pi-kompatibel ist, 3D-gestenerkennungsfähig ist und in Python programmiert werden kann.[[13]](#footnote-14) Bei näherer Betrachtung fällt jedoch die geringe Reichweite von maximal 5 cm ins Auge, welche den Sensor für den alltäglichen Badezimmergebrauch nutzlos macht.

Wenn man sich ansieht, aus welchem Grund dies so ist, kann man erkennen, dass dieser Sensor ebenso wie der MGC3x30 GestIC mit einem elektromagnetischen Feld arbeitet, welches beim Skywriter um einiges schwächer ausgeprägt ist als beim GestIC, was diesen Sensor wesentlich kostengünstiger dastehen lässt (15€ im Vergleich zu 125€).

* + 1. ZX Gesture Detection

Der ZX Gesture Sensor bietet eine Reichweite von 30cm, was als Pluspunkt angerechnet werden kann.[[14]](#footnote-15) Die bessere Reichweite liegt an der Erkennungsart. Nämlich wird hier eine selbstentwickelte und patentierte GestureSense-Technologie verwendet.

Diese ermöglicht es dem Sensor mittels Infrarotstrahlung ausgehend von zwei LEDs rückkommende Strahlung per Mikrocontroller zu interpretieren und so Bewegungen und Abstände zu messen. Dabei werden jedoch nur die x- und z-Position erkannt. Dieser Fakt bringt den Nachteil mit sich, dass Bewegungen nach oben oder unten nicht interpretiert werden. Dies würde bedeuten, dass der Sensor an der Decke angebracht werden müsste, damit die gewollten Gesten erkannt werden. Denn nur dann wären die Achsen so ausgerichtet, dass vor dem Spiegel nach oben/unten oder rechts/links gehende Bewegungen interpretiert werden, da die y-Achse in diesem Fall vom Spiegel wegginge.

Da dies das Gesamtprodukt durch eine hervorstehende Nase an der Oberkante unästhetisch und vor allem unpraktisch machen würde, wurde dieses System nicht weiter in Betracht gezogen.

Es würde zwar auch eine alternative Version des Sensors vom selben Hersteller existieren, welche auch Bewegungen in y-Richtung beinhaltet, diese war aber zum Abfragezeitpunkt zu hochpreisig, um näher darauf einzugehen.

* + 1. Gestenbibliotheken

Mit Gestenbibliotheken zu arbeiten, bringt viele Möglichkeiten mit sich. Man kann Aktivitäten vor dem Spiegel mit einer normalen Kamera aufnehmen und im Rahmen der Fähigkeiten der Bibliothek dann diese interpretieren, wie man sie gerade benötigt. Da von einem vorhergehenden Projekt der Raspberry Pi und die zugehörige NoIR-Kamera zur Verfügung standen, war naheliegend, eine Bibliothek zu finden, mit welcher Python-Programmierung möglich ist.

Eine solche Bibliothek ist bspw. die Open-Source-Bibliothek MooseGesture. Diese hatte sich jedoch dahingehend spezialisiert, Handbewegungen in Mausbewegungen umzuwandeln.[[15]](#footnote-16) Dies funktioniert, indem eindeutige Wischbewegungen in die gewollte Richtung eine Funktion auslasen, welche eine entsprechende Mausbewegung am PC-Bildschirm bewirkt. Diese Bibliothek wurde nicht in Betracht gezogen, da kein Cursor benötigt wurde.

Eine Gestenbibliothek, mit welcher Gesten in Befehle umgewandelt werden können, stellt BoofCV. Diese kommt in diesem Stadium in puncto Funktionalitätsumfang nicht an marktführende Varianten heran, da sie noch sehr jung ist und sich im Grunde noch in der Testphase befindet.[[16]](#footnote-17) Dennoch ist BoofCV bereits dazu in der Lage, Objekte wie Gesichter oder komplette Menschen explizit zu erkennen und diese in ihrer exakten Lage zu definieren. Da jedoch noch keine Handbewegungen erkennbar sind, wurde auch diese Bibliothek nicht weiter in Betracht gezogen.

* 1. **OpenCV**

Auf der Suche nach einer den Anforderungen entsprechenden Gestenbibliothek wurde schnell klar, dass über 90% derartiger Projekte mit der Open Source-Bibliothek OpenCV realisiert wurden. Dementsprechend wurde das Interesse schnell auf diese Gestenbibliothek gelenkt.

* + 1. Beschreibung der Gestenbibliothek

OpenCV steht für Open Computer Vision.[[17]](#footnote-18) Diese Bibliothek zeichnet sich vor allem durch ihre vielen Einsatzmöglichkeiten aus. Dies basiert auf einer breit gefächerten Bibliothek, welche es ermöglicht, jede Art von Körperform programmtechnisch als Gestengeber zu definieren. Daher ist es möglich, Bewegungen von Gesichtern, Händen, ganzen Menschen, Fremdkörpern uvm. für programminterne Befehle zu nutzen.

Das Prinzip von OpenCV ist, einen Körper an einer beliebigen Rückstrahlung zu erkennen, indem das Bildmaterial in Graustufen unterteilt wird und dann die sich ergebenden Konturen mit dem programmtechnisch definierten Muster verglichen werden.

OpenCV besitzt Schnittstellen für C++, C, Python und Java. Da zusätzlich Raspbian eines von vielen unterstützten Systemen ist, fiel die Wahl schlussendlich auf diese Bibliothek und es wurde begonnen, am RPI ein Python-Programm zur Gestensteuerungsrealisierung zu schreiben.

Da OpenCV das Bild in Matrizen unterteilt, um die Unterschiede zum vorhergehenden Bild zu erkennen und so Bewegungen zu detektieren, muss zum Zustandebringen dieser Unterteilung in Python programmintern die Software „NumPy“ inkludiert werden. NumPy ist eine Programmbibliothek, welche durch Datenstrukturen und Funktionen eine einfache Handhabung von Vektoren und den angesprochenen Matrizen ermöglicht.

* + 1. NoIR-Pi-Kamera mit Gesten-Bibliothek

Von Leap Motion konnte mitgenommen werden, dass die Realisierung der Informationsbeschaffung über Handbewegungen mittels RGB-Kamera und/oder eigens dafür konstruierten Sensoren ideal zu sein scheint. Dabei wurden zuerst Überlegungen dahingehend angestellt, wie am unauffälligsten ein Kamera-Modul am Rand des Spiegels angeschlossen werden kann. Dabei kam als erstes die Idee auf, einen Raspberry Pi in Verbindung mit der NoIR-Pi-Kamera einzusetzen.

Dieses Modul war mit 5 Megapixel und einem Infrarot-Sensor dazu in der Lage, Bilder im RGB-Format aufzunehmen und Entfernungen abzuschätzen.[[18]](#footnote-19) Als Erweiterung wurde ein sogenanntes „Fischauge“ organisiert, womit die Erkennungsspannweite wesentlich vergrößert werden konnte.

Um mit diesem Kamera-Modul am RPI arbeiten zu können, musste im Vorfeld ein Treiber für die Videoaufnahme über die NoIR-Kamera installiert werden. Dafür musste im Terminal der Befehl *sudo modprobe bcm2835-v4l2* eingegeben werden. Dieser Vorgang musste bei jedem Neustart des Raspberry Pi wiederholt werden.

In der nächsten Phase musste ein Programm erstellt werden, welches die vier benötigten Gesten erkennen und interpretieren kann. Dafür wurde im Vorfeld eine Bibliothek zur Gestenerkennung benötigt, welche in Verbindung mit Python funktioniert.

* + 1. Gesteninterpretationsprogramm

Von der Installation von OpenCV auf dem Raspberry Pi wurde laut einem im Internet gefundenen Projekt zum Testen der Funktionalität der Software vorgegangen.[[19]](#footnote-20) Dazu wurde der Pseudoprogrammcode umgeschrieben und überprüft, ob es dasselbe Resultat lieferte wie die Vorlage.

Dieses Programm zeigt unter anderem das Echtzeitbild am RPI an und fügte einen rechteckigen Rahmen hinzu, in dem die Handgeste erkannt werden konnte. Dieser Bereich konnte mittels Koordinatenangabe beliebig verschoben und in der Größe variiert werden. Innerhalb dieses Rahmens konnten abhängig von der Anzahl der ausgestreckten Finger diverse Ausgabetexte angezeigt werden.

Nachdem dies wie erwartet funktionierte, wurde begonnen, ein eigenes Programm zu schreiben, dass den Projektanforderungen entsprechend die vier Gesten hin zu den vier Widgets erkennt, interpretiert und in der Folge den Array-Wert des Widgets zurückgibt, sodass das Hintergrundprogramm damit arbeiten kann.

Beim Einlesen in das Python-basierte Arbeiten mit OpenCV wurde schnell klar, dass bei der Programmierung auf viele Funktionen zugegriffen werden muss, dessen Bezeichnungen nachgesehen werden mussten. Diese befinden sich allesamt in der CV2-Bibliothek für Python.

Hierbei wurde mit dem Inkludieren der benötigten Libraries begonnen. Dieser Schritt beinhaltete die vorher angesprochene CV2-Library, mit der die Videoaufnahme begonnen wird und die Handgesten erkannt und interpretiert werden, die NumPy-Library zum Arbeiten mit Matrizen und Vektoren und die Math-Library zur Berechnung von Winkeln und Längen, welche zur Erkennung der Handkonturen benötigt werden.

|  |
| --- |
| import cv2 import numpy as np import math   capture = cv2.VideoCapture(0) sector nr = 0 |

Codeabschnitt 128: Voreinstellungen (gesture\_v4.py)

Des Weiteren ist im Vorfeld nötig, den Zugriff auf die Videoaufnahme zu definieren. Dies wird durch eine Variable ermöglicht, welche mit der CV-eigenen Videoaufnahme vom übergebenen Kamera-Port gleichgesetzt wird. Hinzu kommt eine Variable, die den Array-Index des Sektors angibt, welcher auf Gesten geprüft werden soll.

Erst jetzt kann in die while-Schleife geschritten werden. Diese wird nicht verlassen, ehe nicht die Signalübertragung vom Video-Port zum Python-Programm beendet wurde. In dieser Schleife werden nun jene Funktionen immer wieder aufgerufen, welche für die Erstellung der Form entsprechend der Handkonturen verantwortlich sind. Außerdem werden Form und Position der Hand interpretiert und entsprechend dieser weiter im Code vorgegangen.

|  |
| --- |
| hile capture.isOpened():  for sector\_nr in range(0, 3):   ret, img = capture.read()   ifsector\_nr == 0**:**  cv2.rectangle(img,(0,0),(320,240),(0,255,0),0)  crop\_image = img[0:240,0:320]  elif sector\_nr == 1:  cv2.rectangle(img,(320,0),(640,240),(0,255,0),0)  crop\_image = img[0:240,320:640]  elif sector\_nr == 2:  cv2.rectangle(img,(0,240),(320,480),(0,255,0),0)  crop\_image = img[240:480,0:320]  elif sector\_nr == 3:  cv2.rectangle(img,(320,240),(640,480),(0,255,0),0)  crop\_image = img[240:480,320:640] |

Codeabschnitt 129: Erstellen des Bereichs, in welchem Geste erkennt werden soll (gesture\_v4.py)

Innerhalb dieser Schleife wird in eine for-Schleife gesprungen, welche dazu dient, die Sektoren entsprechend des Array-Index durchzulaufen und auf Aktivitäten zu prüfen. In weiterer Folge werden die Aufnahmen in zwei später benötigten Variablen gespeichert und das Rechteck gezeichnet, in dem die Geste erkannt werden soll. Dabei wird die Position des Rechtecks abhängig vom Sektor-Index angegeben. Die Größe bleibt dabei gleich, um vier Quadranten zu erzielen.

Sobald das Rechteck erstellt ist, kann dazu übergegangen werden, das ankommende Farbbild zu interpretieren. Hierfür ist es zuerst nötig, das Bild in Graustufen zu unterteilen.

|  |
| --- |
| blur = cv2.GaussianBlur(crop\_img, (3,3), 0) hsv = cv2.cvtColor(blur, cv2.COLOR\_BGR2HSV) |

Codeabschnitt 130: Aufteilen des Farbbilds in Graustufen (gesture\_v4.py)

Zur weiteren Bearbeitung des Bildmaterials wird nun das Bild dahingehend unterteilt, dass Stellen ab einem gewissen Helligkeitsgrad als weiß und der Rest als schwarz unterteilt werden. Daraus folgt, dass die im Allgemeinen hellere Hand vom dunkleren Hintergrund unterschieden werden kann.

Die nächste Funktion aktiviert einen Kernel, welcher es ermöglicht, Bewegungen von Körpern zu erkennen und zu interpretieren.

Dies wird dann dazu genutzt, Hintergrundstörungen aus dem Weg zu bekommen und so ein reines Abbild von menschlichen Körperteilen oder Ähnlichfärbigem zu visualisieren.

|  |
| --- |
| mask2 = cv2.inRange(hsv, np.array([2,0,0]), np.array([20,255,255]))  kernel = np.ones((5,5))  dilation = cv2.dilate(mask2, kernel, iterations = 1) erosion = cv2.erode(dilation, kernel, iterations = 1)  filtered = cv2.GaussianBlur(erosion, (3,3), 0) ret\_thresh = cv2.threshold(filtered, 127, 255, 0) |

Codeabschnitt 131: Erstellen einer klaren Unterscheidung zwischen hellen und dunklen Stellen (gesture\_v4.py)

Die Funktionen *Gaussian Blur* und *Threshold* dienen dazu, eine bessere Abgrenzung zwischen weißen und schwarzen Stellen zu erzielen. Das hilft dabei, die Konturen klarer erkennen zu können und somit einfacher die Form der Hand festzustellen.

Das durch diese Funktionen zustande gekommene Bild wird dann zur besseren Einsicht angezeigt. Mit der Funktion *findContours* werden die Konturen von weiß zu schwarz generell gesucht.

|  |
| --- |
| cv2.imshow("Thresholded", thresh)  img, contours, hierarchy = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE) |

Codeabschnitt 132: Anzeigen des schwarz-weißen Resultats (gesture\_v4.py)

Im weiteren Verlauf des Programms stoßt man auf ein sogenanntes Exception-Handling. Dieses besagt, dass wenn keine Konturen erkannt wurden, der Konturen interpretierende Programmbereich übersprungen wird. Falls diese „Ausnahme“ nicht auftreten sollte, geschieht in der Folge zuerst eine Suche nach jenem hellhäutigen Körper, welcher die größte Fläche einnimmt und somit die größte Relevanz hat. Infolgedessen wird über diesen Körper ein Rechteck gestülpt. Weiters wird eine Umhüllende um diese größte erkannte Fläche gezogen.

|  |
| --- |
| try**:**  contour = max(contours, key = lambdax: cv2.contourArea(x))   x,y,w,h = cv2.boundingRect(contour)  cv2.rectangle(crop\_img, (x,y),(x+w,y+h),(0,0,255),0)   hull = cv2.convexHull(contour) |

Codeabschnitt 133: Erstellen einer Umrandung um größte weiße Stelle (gesture\_v4.py)

Anhand der Umhüllenden werden dann im nächsten Schritt Spitzen und Kerben analysiert und in eine eigene Variable gespeichert.

|  |
| --- |
| Drawing = np.zeros(crop\_img.shape,np.uint8) cv2.drawContours(drawing,[contour],-1,(0,255,0),0) cv2.drawContours(drawing,[hull],-1,(0,0,255),0)  hull = cv2.convexHull(contour, returnPoints=False) defects = cv2.convexityDefects(contour,hull)  count\_defects = 0 |

Codeabschnitt 134: Zeichnen der Konturen der sich ergebenden Form (gesture\_v4.py)

Aufgrund der in *defects* gespeicherten Daten kann man nun mittels einiger mathematischer Berechnungen wie Winkel- oder Längenberechnungen feststellen, ob es sich bei dieser Form um eine Hand halten könnte und falls dies der Fall ist, wie viele Finger von dieser Hand ausgestreckt sind.

|  |
| --- |
| for i in range(defects.shape[0]):  s,e,f,d = defects[i,0])  start = tuple(contour[s][0])   end = tuple(contour[e][0])   far = tuple(contour[f][0])   a = math.sqrt((end[0] - start[0])\*\*2 + (end[1] - start[1])\*\*2)   b = math.sqrt((far[0] - start[0])\*\*2 + (far[1] - start[1])\*\*2)   c = math.sqrt((end[0] - far[0])\*\*2 + (end[1] - far[1])\*\*2)   angle = (math.acos((b\*\*2 + c\*\*2 - a\*\*2)/(2\*b\*c))\*180)/3.14   if angle <=90:  count\_defects += 1  cv2.circle(crop\_img,far,1,[0,0,255],-1)   cv2.line(crop\_img,start,end,[0,255,0],2) |

Codeabschnitt 135: Winkelberechnungen zu Erkennung von ausgestreckten Fingern (gesture\_v4.py)

Durch die obigen Berechnungen wird die Kontur rund um den fokussierten Körper erstellt. In der darauffolgenden if-Anweisung werden die ausgestreckten Finger hochgezählt.

|  |
| --- |
| if 5 >= count\_defects >= 3:  if sector\_nr == 0:  cv2.putText(img,"0",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2)  print("0")  elif sector\_nr == 1:  cv2.putText(img,"1",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2)  print("1")  elif sector\_nr == 2:  cv2.putText(img,"2",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2)  print("2")  elif sector\_nr == 3:  cv2.putText(img,"3",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2)  print("3")  else:  cv2.putText(img,"Do nothing",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2)  except:  cv2.putText(img,"Do nothing",(50,50),cv2.FONT\_HERSHEY\_SIMPLEX,2,2) |

Codeabschnitt 136: Ausführen der der Handgesten entsprechenden Ausgabe (gesture\_v4.py)

Die sich ergebende Anzahl wird im Folgenden dazu verwendet, dem Programm nur dann die Erlaubnis zu erteilen, das Widget zu öffnen, sofern zumindest drei Finger an der Hand erkannt wurden. So wird garantiert, dass es sich bei dem erkannten Körper tatsächlich höchstwahrscheinlich um eine menschliche Hand handelt. Je nach Index des Abschnitts, in welchem die Hand hineingewischt wurde, wird dann der Befehl erteilt, das Widget mit dem entsprechenden Index zu öffnen.

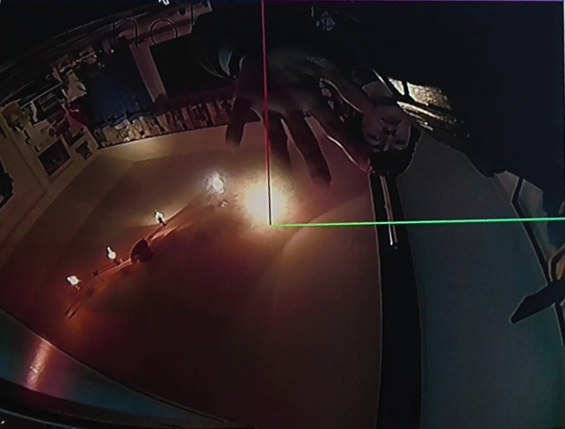


Abbildung 50: Erkennung der Handgeste in OpenCV

Bei Praxistests trat jedoch nicht der gewünschte Erfolg ein. Im Gegenteil, das Programm hatte augenscheinlich Probleme, die Hand im entsprechenden Sektor komplett zu erkennen. Infolgedessen wurde in der Beschreibung von OpenCV nachgesehen, woran das liegen konnte. Dabei wurde erkannt, dass es nicht möglich war, vier Sektoren gleichzeitig anzuzeigen und zu differenzieren.

Daher wurde versucht, dies mittels for-Schleife innerhalb einer endlos-Schleife zu übergehen und in jedem Durchlauf einen anderen Sektor per Rechteck anzusehen. Da diese for-Schleife jedoch ziemlich schnell durchlaufen werden müsste, um regelmäßige Bildaktualisierungen zu garantieren, gerieten wir in eine Sackgasse. Nämlich wurde der fokussierte Sektor so schnell gewechselt (ca. 100ms), dass es dem Interpretationsprogramm nicht möglich war, die Hand und ihre Konturen in allen Details aufzunehmen. Als diese Zeit nach oben hin korrigiert wurde, war das Programm wieder so langsam, dass man die Hand für unbequem lange Zeit in dem entsprechenden Sektor halten musste. Dies war unterm Strich nicht zufriedenstellend genug, um bei OpenCV zu bleiben.

* 1. **APDS-9960**

Nachdem klar war, dass es mittels NoIR-Kamera in Verbindung mit der Gestenbibliothek OpenCV nicht möglich war, die Gestensteuerung optimal aufzubauen, wurde auf der Suche nach der optimalen Realisierungsmöglichkeit ein neuer Weg eingeschlagen. Die Suche entfernte sich vom Gedanken, die Geschehnisse in Front des Spiegels per Kamera-Modul aufzunehmen. Dabei spielte auch das dadurch potenziell hervorgerufene Gefühl der Störung der Privatsphäre durch Überwachung eine wesentliche Rolle. Deshalb wurden Überlegungen angestellt, Aktivitäten anderweitig aufzunehmen. Dabei war ein Infrarotsensor die erste und naheliegendste Idee.

Auf der Suche nach etwas Dementsprechendem wurde klar, dass dieses Schicksal unter anderem auch bereits einige Blogger ereilte. Diese griffen nach der Ernüchterung oftmals auf einen Sensor namens APDS-9960 zurück. Da dieser in den meisten Fällen den gewünschten Erfolg einbrachte, war es nur logisch, diesen Sensor in Betracht zu ziehen.

* + 1. Allgemeine Beschreibung

Von APDS-9960 funktioniert statt mit der Aufnahme von RGB-Bildern via Infrarotschwingungen.[[20]](#footnote-21) Dies bietet auf der einen Seite einen großen Vorteil in puncto Schutz der Privatsphäre, bringt auf der anderen Seite aber den Nachteil mit sich, dass die Reichweite bzw. der Empfang eingeschränkt sind. Die maximale Reichweite wird vom Hersteller mit 10-20cm angegeben und ist nur unter Umständen für unsere Anwendung ausreichend. Auf die Lösung dieses Problems wird später eingegangen.



Abbildung 51: APDS-9960

Ein weiterer Vorteil des APDS-9960 ist, dass er in Verbindung mit Arduino-Boards optimal funktioniert, da der Sensor über C++-Code gesteuert werden kann. Dies spricht für dieses Produkt, da hierfür das im 3. Jahrgang in der schulinternen Werkstätte hergestellte HTL UNO als Schnittstelle verwendet werden kann. Dadurch sinken die Kosten ebenso wie durch den Produktpreis selbst. Dieser liegt mit rund 13€ in einem höchst akzeptablen Bereich.

Weiters auf der Haben-Seite liegt, dass vorprogrammierte Gesten bereits implementiert waren. Unter diesen sechs Gesten waren in gewisser Art und Weise auch die vier Gesten, die für die Realisierung der Gestensteuerung von Nöten waren. Während vier Gesten benötigt wurden, welche von der Mitte ausgehend in die vier Ecken gehen sollten, sind hier unter anderem Gesten in beide horizontalen und beide vertikalen Richtungen vorhanden.

Leider bietet die Bibliothek dieses Sensors keine ernsthafte Möglichkeit, den Horizont hinsichtlich erkennbarer Gesten über die sechs Vorinstallierten hinaus zu erweitern. Die einzige Möglichkeit, dies zu umgehen wäre mittels Auslesen der Bitfolge gewesen. Nach einer eingehenden Analyse mittels Saleae Logic Analyzer wurde erkannt, dass dies einen enormen Aufwand darstellen würde, da die vom Sensor abgegebenen Bitfolgen lang und alles andere als einfach zu entschlüsseln sind. Die vordefinierten Gesten werden jedoch mit einem simplen Trick ohnehin zu den benötigten Wischgesten. Dazu muss der APDS-9960 nur um 45° versetzt am Rahmen des Spiegels fixiert werden.

Weitere wichtige technische Details sind unter anderem die Möglichkeit der automatischen Aktivierung über Distanzverringerung vor dem Sensor und eine 32-Bit-FIFO-Übertragung der Gesteninformationen.[[21]](#footnote-22)

Unterm Strich hatte das Produkt alle Kriterien erfüllt, um Zeit und Geld in die nähere Betrachtung zu investieren und einen Versuch zu starten, diesen Sensor in das Projekt zu implementieren.

* + 1. Verdrahtung am Arduino

Um die Gesten des Infrarotsensors einlesen zu können, musste zuerst in Erfahrung gebracht werden, mit welcher Schnittstelle dieser funktioniert. Beim Durchlesen der Produktbeschreibung des APDS-9960 wurde erkannt, dass dieser per I2C arbeitet. Dabei stellt SCL die Taktleitung und SDA die Datenleitung dar. Dazu kommt die INT-Leitung, welche als Interrupt-Leitung verwendet wird. VCC und GND stehen für die Spannungsversorgung zur Verfügung, ebenso wie VL, welches für die LED-Spannung steht und nicht zur Verwendung kommt, da die verbaute LED nicht beachtet wird.

Die GND-Leitung wurde klarerweise mit einem GND-Pin des UNO-Boards verbunden. Um die VCC-Leitung anschließen zu können, musste zuerst herausgefunden werden, mit welcher Versorgungsspannung der Sensor funktioniert. Diese wurde aus dem Datasheet mit 3,3V entnommen.

Die Leitungen SDA und SCL müssen mit Analog-Eingängen verbunden werden, damit das Board mit dessen Signalen arbeiten kann. Die INT-Leitung soll auf einen digitalen Pin gelegt werden. Dabei ist zu beachten, dass 0 und 1 als RX- und TX-Leitung praktisch wegfallen. Weiters gehört bedacht, dass gewisse Pins nur PWM-Signale einlesen bzw. ausgeben und daher als Interrupt-Pins ungeeignet sind. Zur Wahl stehen darausfolgend also nur mehr wenige Pins.

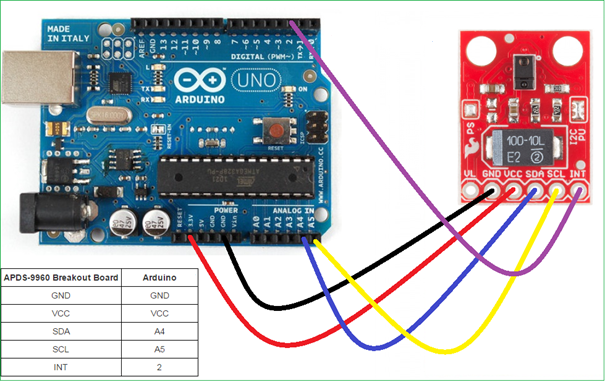


Abbildung 52: Verdrahtung des Arduino UNO mit dem APDS-9960

Wie in der Abbildung zu sehen, wurde Pin 2 bevorzugt. Es könnten jedoch theoretisch auch die Pins 4, 7, 8, 12 und 13 gewählt werden.

* + 1. Gesteninterpretation

Um im Arduino-Programm mit der Ausgabe des Infrarotsensors arbeiten zu können, ist es zuallererst nötig, die von dessen Hersteller SparkFun zur Verfügung gestellte Bibliothek zu integrieren. Um dies bewerkstelligen zu können, führt kein Weg daran vorbei, im Vorfeld die Header-Datei *Wire.h* zu inkludieren.

|  |
| --- |
| #include <Wire.h> #include <SparkFun\_APDS9960.h>  #define APDS9960\_INT 2 |

Codeabschnitt 137: Inkludierungen und Vordefinierungen (Gesture\_detection\_v8.ino)

Weiters muss der Interrupt-Pin des APDS-9960 vordefiniert werden, bevor dessen Initialisierungsprozess begonnen werden kann. Dazu kann zwischen den digitalen Pins des HTL Uno von 1 bis 13 unter bereits zuvor erklärten Einschränkungen gewählt werden. Gewählt wurde, wie in Abbildung 51 zu sehen, Pin 2, da er der in aufsteigender Reihenfolge gesehen erste komplett eigenständige Pin ist, weil die Pins 0 und 1 auch als RX- bzw. TX-Leitung verwendet werden, welche nur in Verbindung zu anderen RX-TX-Ausgängen zum Einsatz kommen sollten.

Im Vorfeld werden einige globale Variablen benötigt:

|  |
| --- |
| SparkFun\_APDS9960 apds = SparkFun\_APDS9960(); int isr\_flag = 0; bool state = false; |
|  |

Codeabschnitt 138: Globale Variablen (Gesture\_detection\_v8.ino)

Die Variable *apds* dient zur Initialisierung des Infrarotsensors im Arduino-Programm. Die *isr\_flag* hilft der *interrruptRoutine,* eine aufgenommene Bewegung zu signalisieren. In der Statusvariable *state* wird die Information darüber gespeichert, ob sich das Java-Programm zum User Interface aktuell in einer Widget-Ansicht oder am Startbildschirm befindet.

In der *interruptRoutine()*-Funktion wird, wie zuvor erwähnt, die *isr\_flag* gesetzt.

|  |
| --- |
| void interruptRoutine() {  isr\_flag = 1; } |

Codeabschnitt 139: interruptRoutine (Gesture\_detection\_v8.ino)

Dadurch wird erkenntlich gemacht, ob eine Geste erkannt wurde. Im folgenden Programm wird diese Funktion so verwendet, dass die Gesteninterpretation ausgeführt wird, wenn in diese Funktion gesprungen wird.

Die *setup()*-Funktion ist diejenige, welche beim Programmstart als erstes aufgerufen wird. In ihr werden alle im Vorfeld nötigen Prozesse ausgeführt.

|  |
| --- |
| void setup() {  pinMode(APDS9960\_INT, INPUT);  Serial.begin(9600);  attachInterrupt(0, interruptRoutine, FALLING);  apds.init();  apds.enableGestureSensor(true); } |
|  |

Codeabschnitt 140: Initialisierungsprozesse (Gesture\_detection\_v8.ino)

Die *pinMode(…)*-Funktion setzt den zuvor auf den digitalen Pin mit der Kennziffer 2 gesetzten Interrupt-Pin als Input-Pin fest.

Daraufhin startet der Initialisierungsprozess des Gestensensors. Bei erfolgreicher Initialisierung startet das Abgreifen der vom Interrupt-Pin kommenden Daten.

Nachdem alle vorhergehenden Prozesse beendet sind, wird die *loop()*-Funktion aufgerufen.

|  |
| --- |
| void loop() {  if (isr\_flag == 1) {  detachInterrupt(0);  handleGesture();  isr\_flag = 0;  attachInterrupt(0, interruptRoutine, FALLING);  } } |

Codeabschnitt 141: Ausführen der Ausgabe-Funktion bei aktivierter Interrupt-flag (Gesture\_detection\_v8.ino)

In dieser schleifenartig agierenden Funktion wird der Gesten-Sensor auf Aktivitäten überprüft. Die if-Anweisung ermöglicht das Aufrufen der *handleGesture()*-Funktion unter dem Umstand, dass der Sensor eine Bewegung wahrnimmt. Ansonsten wird sofort die nächste Überprüfung auf relevante Aktivitäten gestartet.

Wird jedoch tatsächlich eine Aktivität des Sensors erkannt, hat dies zuerst zur Folge, dass die *interruptRoutine*() ausgeführt wird. In weiterer Folge springt das Programm aufgrund der nun gesetzten *isr\_flag* in die *handleGesture()*-Funktion, in welcher die Bewegung entsprechend interpretiert wird. In dieser Zeit ist es nicht möglich, dass der Interrupt erneut ausgelöst wird. Dadurch wird eine Verwirrung innerhalb des Programms vermieden. Erst nach Beendigung der Funktion wird das Interrupt wieder für den Verwender freigegeben.

|  |
| --- |
| void handleGesture() {  if (gestureSensor.isGestureAvailable()) {  switch (gestureSensor.readGesture()) {  case DIR\_UP:  if(state == false) {  Serial.println(2);  state = true;  }  else {  Serial.println(-1);  state = true;  }  break;   case DIR\_DOWN:  if(state == false) {  Serial.println(1);  state = true;  }  break;   case DIR\_LEFT:  if(state == false) {  Serial.println(3);  state = true;  }  break;   case DIR\_RIGHT:  if(state == false) {  Serial.println(0);  state = true;  }  break;   default:  break;  }  } } |
|  |

Codeabschnitt 142: Ausgabe der Befehle entsprechend der benötigten Bedingungen (Gesture\_detection\_v8.ino)

In dieser Funktion wird zu allererst überprüft, ob die Gestenbibliothek von SparkFun es überhaupt ermöglicht, die aktivierte Geste zu interpretieren. Infolgedessen kann die Geste eingelesen werden. Sollte dies gelungen sein, wird der darauffolgende Code ausgeführt. In diesem wird mittels switch-case-Anweisung definiert, was geschehen soll, wenn in die bestimmte Richtung gewischt wurde. Zu diesem Zweck fragt das Programm zuerst in jedem der vier zu interpretierenden Fälle eine Status-Variable ab. Diese enthält die Information darüber, ob sich das Java-Programm aktuell in der Detailansicht einer Anwendung befindet oder nicht. Sollte dies nicht der Fall sein, kann bedenkenlos jegliche der vier Richtungsgesten aufgenommen werden.

Zu sehen ist, dass je nach Wischrichtung der Index des sich hier befindlichen Widgets mittels *Serial.println()* hinausgesendet wird. Diese serielle Ausgabe wird an den Raspberry Pi gesendet und von einem Java-Programm entsprechend interpretiert.

Infolgedessen wird auch die Status-Variable auf *true* gesetzt, da durch die Ausgabe das Java-Programm in die Detailansicht der Anwendung mit dem zutreffenden Index wechselt. Daher wird von diesem Zeitpunkt an nur noch dann eine Ausgabe an den Raspberry Pi weitergegeben, wenn eine Geste nach links unten vonstattengeht. Denn dies wird programmtechnisch als „return“-Geste verstanden. In diesem Szenario wird die Status-Variable wieder auf *false* gesetzt, um kennzuzeichnen, dass das Java-Programm sich nun wieder in der Start-Ansicht befindet und daher alle vier Richtungsgesten durchführbar sind.

Bevor der Sensor in Kombination mit dem Spiegel getestet werden konnte, wurde das Programm verifiziert und mit dem „Serial Monitor“ außerhalb des Gesamtsystems auf den Infrarot-Sensor hochgeladen. Der „Serial Monitor“ wird von der Arduino-Software zur Verfügung gestellt und kann über den Reiter „Tools“ oder das Kürzel *Strg + Shift + M* geöffnet werden. Dabei wurde erkannt, dass das Programm tatsächlich bei entsprechender Eingabe auch die zu erwartende Ausgabe lieferte.

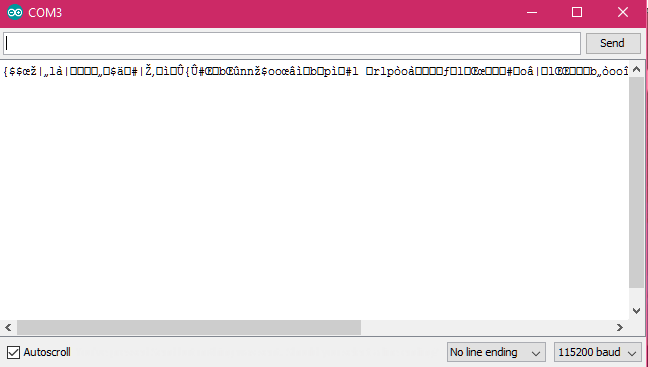
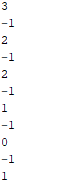
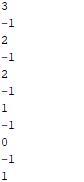


Abbildung 53: Test am „Serial Monitor“

Es ist erkennbar, dass das Programm wie beabsichtigt zuallererst ausschließlich eine Geste in eine beliebige Ecke interpretiert und entsprechend der Position den Array-Index als *Serial.println() ausgibt.* Ist dies ein Mal geschehen, kann nur mehr die Geste in die linke untere Geste eingelesen werden. Diese wird in diesem Fall als „-1“ ausgegeben, um den daraus resultierenden Befehl des Verlassens der Detailansicht vom Fall des Öffnens der sich in der linken unteren Ecke befindlichen Anwendung abzugrenzen.

Zu guter Letzt wurden die Drähte am Sensor angelötet und der Sensor unterhalb des Spiegels verbaut. Bei den Tests mit der finalen Version am Spiegel kam ans Licht, dass die Reichweite auch am tatsächlichen Objekt ähnlich dem oberen Rand der Herstellerangaben bis zu einer Reichweite von 20cm einwandfrei funktioniert und dann ziemlich schnell zu keiner Reaktion übergeht.

Die Gesten wurden schnell interpretiert, sofern es die Internetverbindung zuließ, welche allerdings wegen dem geringfügigen Datenstrom kein allzu großes Problem darstellen konnte. Die Durchführung erfolgte auch nach zwei Stunden andauernden Tests immerzu fehlerfrei.



Abbildung 54: Gestensensor im Einsatz

In der obigen Abbildung wird das Wischen der Hand über den Infrarotsensor symbolisch dargestellt, um den Wechsel in die Detailansicht eines Widgets zu signalisieren.

* + 1. GP2Y0A02YK0F-Infrarotsensor als Erweiterung

Jedoch blieb ein kleiner Negativpunkt mit dem APDS-9960 über. Eine Aufnahmemöglichkeit mit einer Erkennungsdistanz von maximal 20cm ist nicht unbedingt ideal in den Alltag integrierbar. Zwar sollte die Reichweite auch nicht allzu groß sein, da erkennungstechnische Missverständnisse damit einhergehen könnten, aber immerhin ist es im Sinne des Nutzer-Komforts, wenn er nicht mehr tun muss, als sich vor sein Waschbecken zu stellen und mit der Hand einmal kurz über den Spiegel zu wischen, um Details zu sehen, anstatt den Arm unnötig in Richtung Spiegel zu strecken.

Bei der Suche nach einer Lösung des Problems wurde ersichtlich, dass sich dieses Problem auch anderen stellte und viele dies mit einem zusätzlichen Infrarotsensor lösen konnten. Gemeint war der GP2Y0A02YK0F, welcher die Reichweite von bis zu 20cm auf bis zu 50cm vergrößern konnte. [[22]](#footnote-23) Damit wäre ein idealer Bereich erreicht.

Allerdings ist der Bereich von 20-50cm Distanz für die Wischgestenerkennung relativ unbrauchbar, da hier nur auf Distanzveränderungen reagiert werden kann. Infolgedessen wurde überlegt, diesen Sensor zum Entschwinden aus der Detailansicht herzunehmen. Dies wurde jedoch ebenfalls verworfen, da es nicht für gut befunden wurde, einen zweiten Sensor einzubauen. Stattdessen wurde beschlossen, die Wischgeste in die linke untere Ecke neben der Funktion des Öffnens der sich in dieser Ecke befindlichen Anwendung ebenso als „Exit“-Geste zu verwenden, falls der Anwender sich in der Detailansicht befindet.

1. **iOS-App**

Da die Verwendung von Tastatur und Maus bei einem Smart-Mirror eher wenig Sinn macht, wurde zur Steuerung bzw. zum Setzen von Einstellungen eine Smartphone-App für iOS entwickelt. Die Programmiersprache der App ist zur Gänze Swift.

* 1. **Storyboards**

Der Storyboard-Editor kann verschiedenste Ansichten, auch *ViewController* genannt, grafisch als Gesamtbild darstellen. Diese sind in folgendem Bild ersichtlich:

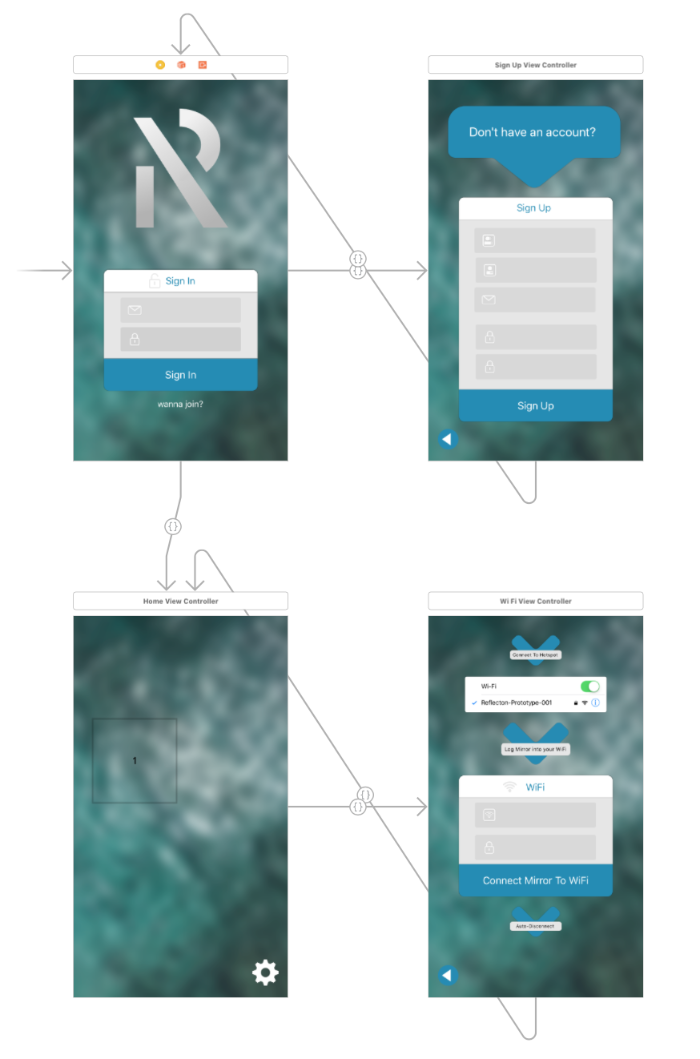


Abbildung 55: Storyboard

Die zu diesen Ansichten zugehörigen Klassen heißen *SignInViewController*, *SignUpViewController*, *HomeViewController* und *WiFiViewController*. Außerdem sind zwischen Ihnen die Segues bzw. Übergänge ersichtlich.

* 1. **Workflow**

Der Workflow von Reflecton in Verbindung mit der zugehörigen iOS-App sieht wie folgt aus:

1. Zuerst muss sich jeder Anwender im *SignUpViewController* ein Benutzerkonto erstellen. Dabei werden die eingegebenen Parameter an das PHP-Interface am externen Ubuntu-Server übertragen. Diese Schnittstelle speichert die Daten schlussendlich in der Datenbank.
2. Folglich kann sich der Benutzer im *SignInViewController* mithilfe der Eingabe von E-Mail und Passwort anmelden.
3. Nun können im *HomeViewController* bereits Widgets verschoben werden. Dies hat allerdings noch keine Auswirkung, da der Reflecton-Spiegel noch nicht mit dem Internet verbunden ist.
4. Nachdem der Spiegel mit einer Stromversorgung verbunden wurde, sendet er einen WLAN-Hotspot aus. Mit jenem Hotspot muss sich manuell mit dem iPhone verbunden werden. Nach einem Wechsel auf den *WiFiViewController* werden bei eingegebener SSID und Passkey, die Parameter *email*, *ssid* und *psk* an das PHP-Interface am Pi übertragen (IP-Adresse: *192.168.50.5*). Dieses PHP-Skript wählt, wie in einem anderen Kapitel genauer beschrieben, den Raspberry Pi in das gewünschte Heimnetzwerk ein. Unter anderem speichert dieses Skript den Parameter *email* in eine bestimmte Textdatei, sodass das Java-UI-Programm den Parameter an der Datenbank als Schlüssel verwenden kann.
5. Nun zeigt auch das Verschieben von Widgets im *HomeViewController* seine Wirkung. Bewegt man ein bestimmtes Widget beispielsweise von links oben nach rechts unten, so ist dieselbe Bewegung am Reflecton-Spiegel festzustellen.
6. Damit auch noch E-Mail- und Kalender-Widget ihren Sinn erfahren, kann bei einem Klick auf eines der beiden Widgets entweder ein E-Mail-Konto bzw. ein Kalender-Konto hinzugefügt werden. Die gewünschten Nachrichten bzw. Termine werden nun auch am Spiegel dargestellt.
   1. **Struktur**

Im unten dargestellten Bild ist die Struktur unserer App ersichtlich. Nun werden die darin befindlichen Dateien besprochen.

|  |  |
| --- | --- |
| Abbildung 56: Struktur | Die Datei *Config.swift* beinhaltet nur eine Sammlung zweier IP-Adressen.  Das File *WaterfallCollection-ViewLayout.swift* von Nicholas Tau hingegen ist für die Animation beim Verschieben der Widgets verantwortlich.[[23]](#footnote-24)  Nebenbei dienen die Datei *Text-CollectionViewLayout.swift*, die *Segue*-Dateien, sowie die *DesignableTextField.swift*-Datei dem Design der App.  Das Herzstück des Programmes liegt in den vier *ViewController*-Klassen, welche die Logik hinter den Ansichten steuern.  Die im Programm notwendigen Bilder werden im Ordner *Assets.xcassets* gespeichert. |

* 1. **Konfigurationsdatei**

Im *Config*-Ordner befindet sich die einzige Konfigurationsdatei. Diese hört auf den Namen *Config.swift* und beinhaltet lediglich zwei IP-Adressen.

|  |
| --- |
| **import** Foundation **let** ipAddressServer = **"173.212.224.225"****let** ipAddressMirror = **"192.168.50.5"** |

Codeabschnitt 143: Vordefinitionen und Konfigurationen (Config.swift)

Bei den Adressen handelt es sich um die des externen Ubuntu-Apache-Servers (notwendig für PHP-Interface-Server), sowie des lokalen Apache-Servers am Pi (notwendig für PHP-Interface-Pi).

* 1. **Designspezifische Dateien**

Diese Dateien sind für die Funktionalität nicht zwingend notwendig. Trotzdem verleihen sie der iOS-App eine professionelle Note. Als designspezifisch sind die Dateien *Water-fallCollectionViewLayout.swift*, *TextCollectionViewCell.swift*, *SegueToLeft.swift*, *SegueToRight.swift*, *SegueToBottom.swift* und *DesignableTextField.swift* zu betrachten.

* + 1. Reflecton/Layouts/WaterfallCollectionViewLayout.swift

Bei dieser Klasse handelt es sich lediglich um einen Code von Nicholas Tau.[[24]](#footnote-25) Dieser kann von Github heruntergeladen werden und dient in unserem Programm zur Animation von Zell- bzw. Widgets-Verschiebungen.

* + 1. Reflecton/Views/TextCollectionViewCell.swift

Eine *UICollectionView* ist eine Sammelansicht von verschiedenen Zellen. Die Klasse *TextCollectionViewCell* erbt von *UICollectionViewCell* und erweitert diese Klasse um ein mögliches Textfeld innerhalb der Zelle bzw. Kachel.

|  |
| --- |
| **import** UIKit  **class** TextCollectionViewCell: UICollectionViewCell {  **@IBOutlet weak var** textLabel: UILabel! } |

Codeabschnitt 144: CollectionView mit Text (TextCollectionViewCell.swift)

Dies ist bei der Variablendeklaration im obigen Programmcode ersichtlich. Die Spezifikation *@IBOutlet* ist hier notwendig, da eine Verbindung mit dem Storyboard verwendet wird.

|  |  |
| --- | --- |
|  |  |

Abbildung 57: Benutzerdefinierte Klasse

Mit einem Klick auf die Zelle im Storyboard-Editor kann die *Custom Class* im rechten Seitenmenü eingetragen werden.

* + 1. Reflecton/CustomSegues/SegueToLeft.swift

In unserer Applikation werden vier verschiedene Ansichten bzw. *ViewController* verwendet. Um die Übergänge zwischen den Ansichten zu gestalten, wurden die Klassen *SegueToLeft*, *SegueToRight* und *SegueToBottom* implementiert. Da bei allen dreien nur der Aufruf der Funktion *CGAffineTransform(...)* unterschiedlich ist, wird nur die Datei *SegueToLeft.swift* abgebildet.

|  |
| --- |
| **import** UIKit  **class** SegueToLeft: UIStoryboardSegue {  **override func** perform() {  **let** src = **self**.source  **let** dst = **self**.destination   src.view.superview?.insertSubview(dst.view, aboveSubview: src.view)  dst.view.transform = CGAffineTransform(translationX: -src.view.frame.size.width,   y: 0)  UIView.animate(withDuration: 0.25, delay: 0.0, options:   UIViewAnimationOptions.curveEaseInOut, animations: {  dst.view.transform = CGAffineTransform(translationX: 0, y: 0)  }) { (finished) **in** src.present(dst, animated: **false**, completion: **nil**)  }  } } |

Codeabschnitt 145: Übergang von rechts nach links (SegueToLeft.swift)

Die oben abgebildete Klasse ermöglicht einen Übergang von rechts nach links. Dabei werden verschiedenste Funktionen aufgerufen. Das Herzstück der Klasse stellt die Funktion *UIView.animate(...)* dar. Beim Übergang zwischen zwei *ViewController* sieht es schlussendlich so aus, als wäre der *Quell-ViewController* rechts vom *Ziel-ViewController* angeordnet.

|  |  |
| --- | --- |
|  |  |

Abbildung 58: Benutzerdefinierte Klasse

Um den benutzerdefinierten Übergang auch verwenden zu können, muss beim Übergang *SegueToLeft* als *Custom Class* ausgewählt werden.

* + 1. Reflecton/CustomElements/DesignableTextField.swift

Damit die Textfelder innerhalb der App auch beliebig gestaltet werden können, wurde die Klasse *DesignableTextField* ins Leben gerufen.

|  |
| --- |
| **import** UIKit  **@IBDesignable class** DesignableTextField: UITextField {  **@IBInspectable var** leftImage: UIImage? {  **didSet** {  **self**.updateView()  }  }    **@IBInspectable var** leftPadding: CGFloat = 5 {  **didSet** {  **self**.updateView()  }  }   **@IBInspectable var** cornerRadius: CGFloat = 0 {  **didSet** {  **self**.layer.cornerRadius = cornerRadius  }  }    **func** updateView() {  **if let** image = leftImage {  **self**.leftViewMode = .always  **let** imageView = UIImageView(frame: CGRect(x: leftPadding, y: 0, width: 40,   height: 40))  imageView.image = image  imageView.tintColor = tintColor  **var** width = leftPadding + 30    **if** borderStyle == UITextBorderStyle.none || borderStyle ==   UITextBorderStyle.line {width = width + 20;  }   **let** view = UIView(frame: CGRect(x: 0, y: 0, width: width, height: 40))  view.addSubview(imageView)  **self**.leftView = view  }  **else** {  **self**.leftViewMode = .never  }    **self**.attributedPlaceholder = NSAttributedString(string: placeholder!,   attributes: [NSAttributedStringKey.foregroundColor : tintColor!])  } } |

Codeabschnitt 146: Benutzerdefiniertes Textfeld (DesignableTextField.swift)

Diese benutzerdefinierte Klasse (*Custom Class*) ermöglicht das Einstellen eines Symbolbildes an der linken Seite, den Einzug des Bildes von links, sowie den Radius an den Ecken des Textfeldes. Die eben genannten Parameter können im *Storyboard*-Editor umgestellt werden. Dafür muss als *Custom Class* die Klasse *DesignableTextField* ausgewählt werden.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Abbildung 59: Implementierung des benutzerdefinierten Textfeldes

Bei jeder Veränderung von Bild und Bildeinzug wird die Funktion *updateView()* aufgerufen. Jene Funktion aktualisiert die Ansicht auf die geänderten Werte.

* 1. **Ansichten**

Die vier Ansichten werden mithilfe von Klassen, welche von *ViewController* erben, dargestellt. Es handelt sich hierbei um die Klassen *SignUpViewController*, *SignInViewController*, *HomeViewController* und *WiFiViewController*. Da die eben genannten Klassen die längsten des Projektes sind, werden all ihre Funktionen einzeln abgebildet.

* + 1. Reflecton/ViewControllers/SignUpViewController.swift

Der *SignUpViewController* wird zum Erstellen eines Kontos verwendet. Dabei müssen Vorname, Nachname, E-Mail, Passwort und Passwortbestätigung eingegeben werden.

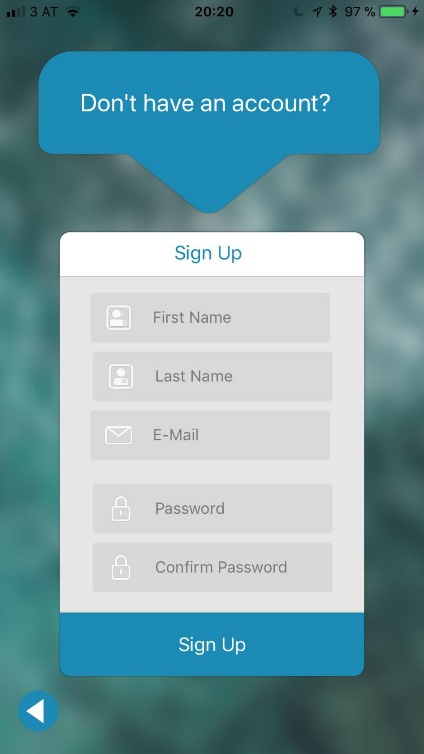


Abbildung 60: Erstellen eines Kontos

Mit einem Klick auf den Button wird ein PHP-Skript am externen Ubuntu-Server aufgerufen und die angegebenen Daten in die Datenbank eingetragen.

Membervariablen

Die Membervariablen belaufen sich alle auf den Typ *UITextField* und haben die Spezifikation @IBOutlet vorangestellt. Diese ist notwendig, um den Code mit dem Textfeld im Storyboard zu verbinden. Die Deklaration wird automatisch ausgeführt,  
wenn das Textfeld mit gedrückter *CTRL*-Taste vom Storyboard in die Klasse *SignUpViewController* gezogen wird.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| *textfieldFirstname* | Textfeld zum Eingeben des Vornamens |
| *textfieldLastname* | Textfeld zum Eingeben des Nachnamens |
| *textfieldEmail* | Textfeld zum Eingeben der E-Mail |
| *textfieldPassword* | Textfeld zum Eingeben des Passworts |
| *textfieldPasswordRepeat* | Textfeld zum Eingeben der Passwortbestätigung |

Funktionen

Die unten gelisteten Funktionen sind notwendige Bestandteile der zur Anzeige verantwortlichen Klasse *SignUpViewController*.

Die Funktion *viewDidLoad()* wird aufgerufen, sobald der komplette *ViewController* in den Memory-Speicher geladen wurde.

|  |
| --- |
| **override func** viewDidLoad() {  **super**.viewDidLoad()  textfieldFirstname.delegate = **self** textfieldLastname.delegate = **self** textfieldEmail.delegate = **self** textfieldPassword.delegate = **self** textfieldPasswordRepeat.delegate = **self** NotificationCenter.default.addObserver(**self**,  selector: **#selector**(SignUpViewController.keyboardWillShow(sender:)),  name: NSNotification.Name.UIKeyboardWillShow, object: **nil**)  NotificationCenter.default.addObserver(**self**,  selector: **#selector**(SignUpViewController.keyboardWillHide(sender:)),  name: NSNotification.Name.UIKeyboardWillHide, object: **nil**) } |

Codeabschnitt 147: ViewController in Memory geladen (SignUpViewController.swift)

Das Delegat (engl. delegate) muss bei jedem einzelnen Textfeld gesetzt werden. Somit weiß das Textfeld über den zugehörigen *ViewController* Bescheid. Die zwei weiteren Funktionsaufrufe mit *addObserver(...)* werden zum Zeigen bzw. Verstecken der virtuellen Tastatur benötigt.

Die von Swift selbsterstellte Funktion *didReceiveMemoryWarning()* wird aufgerufen, wenn ein Problem mit dem Memory-Speicher vorliegt.

|  |
| --- |
| **override func** didReceiveMemoryWarning() {  **super**.didReceiveMemoryWarning() } |

Codeabschnitt 148: Memory-Problemwarnung (SignUpViewController.swift)

Innerhalb des Methodenblocks wird die Funktion *didReceiveMemoryWarning()* der Superklasse *ViewController* aufgerufen.

Die Deklaration der unten abgebildeten Funktion *buttonSignUp(...)* wird automatisch ausgeführt, sobald der *SignUp*-Button mit gedrückter *CTRL*-Taste vom Storyboard in die Klasse *SignUpViewController* gezogen wird.

|  |
| --- |
| **@IBAction func** buttonSignUp(\_ sender: UIButton) {  **let** urlInterface = URL(string: **"http://**\(ipAddressServer)**/Reflecton/api/signup.php"**)  **let** firstname = textfieldFirstname.text  **let** lastname = textfieldLastname.text  **let** email = textfieldEmail.text  **let** password = textfieldPassword.text  **let** passwordRepeat = textfieldPasswordRepeat.text   **var** data = **""** data = data + **"<widget>email</widget>"** data = data + **"<widget>calendar</widget>"** data = data + **"<widget>weather</widget>"** data = data + **"<widget>clock</widget>"   if** !firstname!.isEmpty && !lastname!.isEmpty && !email!.isEmpty &&   !password!.isEmpty && password! == passwordRepeat! {  **let** postParameters = **"first\_name="** + firstname! + **"&last\_name="**+lastname!  + **"&email="**+email! + **"&password="**+password! + **"&data="**+data   **var** request = URLRequest(url: urlInterface!)  request.httpMethod = **"POST"** request.httpBody = postParameters.data(using: .utf8)   **let** task = URLSession.shared.dataTask(with: request) { data, response, error **in  guard let** data = data, error == **nil else** {  print(**"error=**\(String(describing: error))**"**)  **return** }  **if let** httpStatus = response **as**? HTTPURLResponse, httpStatus.statusCode !=   200 {  print(**"statusCode should be 200, but is** \(httpStatus.statusCode)**"**)  print(**"response =** \(String(describing: response))**"**)  }  **let** responseString = String(data: data, encoding: .utf8)  **self**.switchViewController(sender, responseString: responseString!)  }  task.resume()  }   **else** {  **let** alertController = UIAlertController(title: **"Reflecton Sign Up"**,  message: **"The passwords might not fit to each other, or   a required field is missing!"**,  preferredStyle: UIAlertControllerStyle.alert)  alertController.addAction(UIAlertAction(title: **"OK"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  } } |

Codeabschnitt 149: Aktion bei SignUp-Button (SignUpViewController.swift)

Sobald der auf deutsch übersetzte Konto-Erstellen-Knopf gedrückt wird, wird die oben dargestellte Funktion aufgerufen. Innerhalb des Methodenblocks werden zuerst alle Werte der Textfelder ausgelesen. Folglich wird die *data*-Variable mit dem Standardschema der Datenbank befüllt.

|  |
| --- |
| <widget>email</widget>   <widget>calendar</widget>  <widget>weather</widget>  <widget>clock</widget>" |

Codeabschnitt 150: Anordnung der Widgets

Danach folgt der wichtigste Teil der Methode. Es werden die eingegebenen Daten plus der *data*-Variable an die PHP-Schnittstelle des externen Ubuntu-Servers übertragen. Bedingung dafür ist, dass alle Textfelder befüllt sind. Ist diese Bedingung nicht erfüllt, wird eine Warnmeldung angezeigt. Die Übertragung funktioniert mithilfe des HTTP-Post-Request-Prinzips. Als Resümee der Übertragung wird die Variable *responseString* generiert. Verlief die Übertragung erfolgreich (*responseString*="1"), so wird der *ViewController* mithilfe der Funktion *switchViewController*(...) gewechselt. Dabei ist es wichtig, den finalen *responseString* auch an die eben genannte Methode zu übergeben.

Neben dem *SignUp*-Button gibt es auch noch den Back-Button, mit welchem man auf den *SignInViewController* zurückkommt.

|  |
| --- |
| **@IBAction func** buttonBack(\_ sender: UIButton) {  **self**.performSegue(withIdentifier: **"SegueToLeft"**, sender: sender) } |

Codeabschnitt 151: Aktion bei Back-Button (SignUpViewController.swift)

Innerhalb der Funktion wird einfach nur ein Übergang mithilfe der Klasse *SegueToLeft* ausgeführt.

Treten im Zuge des Anmeldevorganges keine Probleme auf, so wird die Funktion *switchViewController(...)* zum Wechseln auf eine andere Ansicht bzw. *ViewController* benötigt.

|  |
| --- |
| **func** switchViewController(\_ sender: UIButton, responseString :String?) {  **let** responseInt = Int(responseString!)   DispatchQueue.main.async(execute: {  **if** responseInt! == 1 {  **let** alertController = UIAlertController(title: **"Reflecton Sign Up"**,  message: **"You are now a member of pinnovations."**,   preferredStyle: UIAlertControllerStyle.alert)  **let** action = UIAlertAction(title: **"OK"**, style: .default) { (action)->Void **in  self**.performSegue(withIdentifier: **"SegueToLeft"**, sender: sender)  }  alertController.addAction(action)  **self**.present(alertController, animated: **true**, completion: **nil**)  }  **else** {  **let** alertController = UIAlertController(title: **"Reflecton Sign Up"**,   message: **"E-Mail already used!"**,  preferredStyle: UIAlertControllerStyle.alert)  alertController.addAction(UIAlertAction(title: **"OK"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  }  }) }  **override func** prepare(for segue: UIStoryboardSegue, sender: **Any**?) {  \_ = segue.destination **as**! SignInViewController } |

Codeabschnitt 152: Wechseln auf andere Ansicht (SignUpViewController.swift)

Wenn die Antwort des Servers *responseInt* der Zahl 1 entspricht, dann wird der Übergang *SegueToLeft* ausgeführt. Der Identifier *SegueToLeft* muss im Storyboard bei einem Custom-Segue auch gesetzt werden. Entspricht die Antwort des Servers *responseInt* nicht der Zahl 1, so ist ein Fehler seitens des Servers aufgetreten und es muss eine Fehlermeldung angezeigt werden. Außerdem muss die Funktion *prepare(...)* implementiert werden. Diese Methode dient zur Vorbereitung des Überganges. Das Ziel ist immer der *SignInViewController*, egal ob die Benutzererstellung erfolgreich war, oder der Benutzer sie mit dem *buttonBack* vorzeitig abgebrochen hat.

Bei einem Klick auf ein Textfeld soll die virtuelle Tastatur von iOS angezeigt werden. Klickt der Benutzer dann auf eine andere Stelle als auf die Tastatur oder Textfeld, dann soll die virtuelle Tastatur wieder ausgeblendet werden. Für dies sind die vier unten abgebildeten Funktionen verantwortlich.

|  |
| --- |
| **func** textFieldShouldReturn(\_ textField: UITextField) -> Bool {  textField.resignFirstResponder()  **return true** }  **override func** touchesBegan(\_: Set<UITouch>, with: UIEvent?) {  textfieldFirstname.resignFirstResponder()  textfieldLastname.resignFirstResponder()  textfieldEmail.resignFirstResponder()  textfieldPassword.resignFirstResponder()  textfieldPasswordRepeat.resignFirstResponder()  **self**.view.endEditing(**true**) }  **@objc func** keyboardWillShow(sender: NSNotification) {**self**.view.frame.origin.y = -160 }  **@objc func** keyboardWillHide(sender: NSNotification) {  **self**.view.frame.origin.y = 0 } |

Codeabschnitt 153: Anzeigen der virtuellen Tastatur (SignUpViewController.swift)

Die Funktion *textFieldShouldReturn(...)* sorgt dafür, dass die virtuelle Tastatur bei einem Klick auf *Return* verschwindet. Um die ganzen Textfeld-Antwortgeber zu quittieren, wird die Methode *resignFirstResponder()* in *touchesBegan(...)* aufgerufen. Die zwei Funktionen *keyboardWillShow* und *keyboardWillHide* sind für die Verschiebung der Ansicht verantwortlich.

* + 1. Reflecton/ViewControllers/SignInViewController.swift

Der *SignInViewController* wird zum Anmelden an einem bestehenden Konto verwendet. Dabei muss die E-Mail-Adresse und das Passwort eingegeben werden. Wenn der Benutzer aber noch kein Konto hat, dann ist die Erstellung mit einem Klick auf „wanna join?“ möglich.

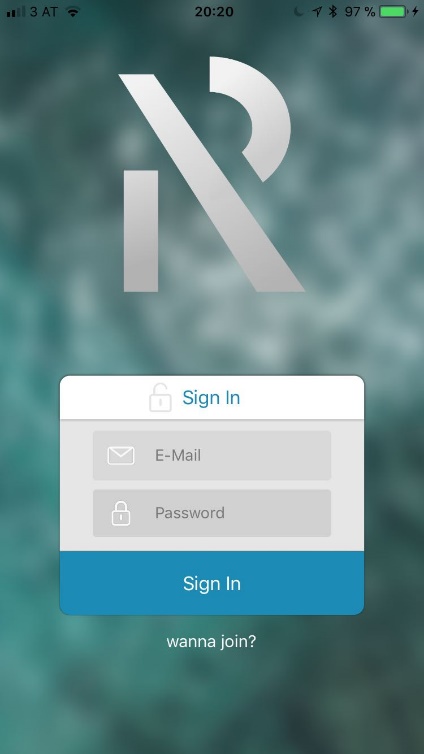


Abbildung 61: Anmelden

Wenn der blaue *SignIn*-Button gedrückt wird, wird ein PHP-Skript am externen Server aufgerufen und die Eingabe abgehandelt.

Membervariablen

Die Membervariablen haben den Typ *UITextField*! und die notwendige Spezifikation *@IBOutlet* vorangestellt. Durch sie ist es möglich, den Code mit dem Storyboard zu verbinden.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| textfieldEmail | Textfeld zum Eingeben der E-Mail |
| textfieldPassword | Textfeld zum Eingeben des Passworts |

Funktionen

Die unten gelisteten Funktionen sind notwendige Bestandteile der Anmeldeansicht *SignInViewController*.

Die Funktion *viewDidLoad()* wird aufgerufen, sobald der komplette *ViewController* in den Memory-Speicher geladen wurde.

|  |
| --- |
| **override func** viewDidLoad() {  **super**.viewDidLoad()  textfieldEmail.delegate = **self** textfieldPassword.delegate = **self** NotificationCenter.default.addObserver(**self**, selector:   **#selector**(SignInViewController.keyboardWillShow(sender:)), name:   NSNotification.Name.UIKeyboardWillShow, object: **nil**)  NotificationCenter.default.addObserver(**self**, selector:   **#selector**(SignInViewController.keyboardWillHide(sender:)), name: aNSNotification.Name.UIKeyboardWillHide, object: **nil**) } |

Codeabschnitt 154: ViewController in Memory geladen (SignInViewController.swift)

Das Delegat (engl. delegate) muss bei jedem einzelnen Textfeld gesetzt werden. Somit weiß das Textfeld über den zugehörigen *ViewController* Bescheid. Die zwei weiteren Funktionsaufrufe mit *addObserver(...)* werden zum Zeigen bzw. Verstecken der virtuellen Tastatur benötigt.

Die von Swift selbsterstellte Funktion *didReceiveMemoryWarning()* wird aufgerufen, wenn ein Problem mit dem Memory-Speicher vorliegt.

|  |
| --- |
| **override func** didReceiveMemoryWarning() {  **super**.didReceiveMemoryWarning() } |

Codeabschnitt 155: Memory-Problemwarnung (SignInViewController.swift)

Innerhalb des Methodenblocks wird die Funktion *didReceiveMemoryWarning()* der Superklasse *ViewController* aufgerufen.

Die Funktion *buttonSignIn(...)* wird automatisch ausgeführt, sobald der *buttonSignIn* gedrückt wurde. Außerdem wird wieder eine Verbindung zwischen *Storyboard* und Code benötigt.

|  |
| --- |
| **@IBAction func** buttonSignIn(\_ sender: UIButton) {  **let** urlInterface = URL(string: **"http://**\(ipAddressServer)**/Reflecton/api/signin.php"**)  **let** email = textfieldEmail.text  **let** password = textfieldPassword.text   **if** !email!.isEmpty && !password!.isEmpty {  **let** postParameters = **"email="** + email! + **"&password="** + password!  **var** request = URLRequest(url: urlInterface!)  request.httpMethod = **"POST"** request.httpBody = postParameters.data(using: .utf8)   **let** task = URLSession.shared.dataTask(with: request) { data, response, error **in  guard let** data = data, error == **nil else** {  print(**"error=**\(String(describing: error))**"**)  **return** }  **if let** httpStatus = response **as**? HTTPURLResponse, httpStatus.statusCode !=   200 {  print(**"statusCode should be 200, but is** \(httpStatus.statusCode)**"**)  print(**"response =** \(String(describing: response))**"**)  }  **let** responseString = String(data: data, encoding: .utf8)  **self**.switchViewController(sender, responseString: responseString!)  }  task.resume()  }  **else** {  **let** alertController = UIAlertController(title: **"Reflecton Sign In"**,   message: **"Please check your entries!"**,  preferredStyle: UIAlertControllerStyle.alert)  alertController.addAction(UIAlertAction(title: **"OK"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  } } |

Codeabschnitt 156: Aktion bei SignIn-Button (SignInViewController.swift)

Nach dem Drücken des Knopfes werden die Parameter an das PHP-Interface übertragen. Dieses sendet dann eine 1 zurück, wenn das jeweilige Konto korrekt gefunden wurde. Eine Zahl ungleich 1 besagt einen Fehler wie z.B. ein falsches Passwort. Dabei wird natürlich eine Fehlermeldung mithilfe eines *UIAlertControllers* angezeigt.

Des Weiteren gibt es noch einen weiteren Knopf namens *buttonJoin*. Wird dieser gedrückt, so wird die Funktion *buttonJoin(...)* ausgeführt.

|  |
| --- |
| **@IBAction func** buttonJoin(\_ sender: UIButton) {  **self**.performSegue(withIdentifier: **"SegueToRight"**, sender: sender) } |

Codeabschnitt 157: Aktion bei Join-Button (SignInViewController.swift)

Hier wird einfach ein Übergang nach rechts, also vom *SignInViewController* zum *SignUpViewController* ausgeführt.

Treten im Zuge des Anmeldevorganges keine Probleme auf, so wird die Funktion *switchViewController(...)* zum Übergang nach unten (*SegueToBottom*) gebraucht.

|  |
| --- |
| **func** switchViewController(\_ sender: UIButton, responseString :String?) {  **let** responseInt = Int(responseString!)   DispatchQueue.main.async(execute: {  **if** responseInt! == 1 {  **self**.performSegue(withIdentifier: **"SegueToBottom"**, sender: sender)  }  **else** {  **let** alertController = UIAlertController(title: **"Reflecton Sign In"**,  message: **"E-Mail and password do not match to each other!"**,   preferredStyle: UIAlertControllerStyle.alert)  alertController.addAction(UIAlertAction(title: **"OK"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  }  }) }  **override func** prepare(for segue: UIStoryboardSegue, sender: **Any**?) {  **if** segue.identifier == **"SegueToBottom"** {  **let** nextViewController = segue.destination **as**! HomeViewController  nextViewController.email = **self**.textfieldEmail.text!  }  **else if** segue.identifier == **"SegueToRight"** {  \_ = segue.destination **as**! SignUpViewController  } } |

Codeabschnitt 158: Wechseln zu Hauptansicht (SignInViewController.swift)

Wenn die Antwort des Servers *responseInt* positiv ist, also der Zahl 1 entspricht, dann wird der Übergang *SegueToBottom* ausgeführt. Sollte das nicht der Fall sein, so wird eine Fehlermeldung angezeigt. Außerdem wird noch die Funktion *prepare(...)* benötigt. Diese Funktion gilt als notwendige Vorbereitung der Übergänge. Wenn der Identifier *SegueToBottom* ist, dann wird der *HomeViewController* aufgerufen und dessen Membervariable *email* initialisiert. Dies wird, wie gesagt, bei einer erfolgreichen Anmeldung benötigt. Wenn der Identifier des Überganges *SegueToRight* ist, dann wird der *SignUpViewController* aufgerufen. Dieser Übergang wird bei einem Klick auf den „Wanna join?“-Knopf benötigt.

Bei einem Klick auf ein Textfeld soll die virtuelle Tastatur von iOS angezeigt werden.

Klickt der Benutzer dann auf eine andere Stelle als auf die Tastatur oder Textfeld, dann soll die virtuelle Tastatur wieder ausgeblendet werden. Für dies sind die vier unten abgebildeten Funktionen verantwortlich.

|  |
| --- |
| **func** textFieldShouldReturn(\_ textField: UITextField) -> Bool {  textField.resignFirstResponder()  **return true** } **override func** touchesBegan(\_: Set<UITouch>, with: UIEvent?) {  **self**.textfieldEmail.resignFirstResponder()  **self**.textfieldPassword.resignFirstResponder()  **self**.view.endEditing(**true**) } **@objc func** keyboardWillShow(sender: NSNotification) {  **self**.view.frame.origin.y = -150 } **@objc func** keyboardWillHide(sender: NSNotification) {  **self**.view.frame.origin.y = 0 } |

Codeabschnitt 159: Anzeigen der virtuellen Tastatur (SignInViewController.swift)

Die Funktion *textFieldShouldReturn(...)* sorgt dafür, dass die virtuelle Tastatur bei einem Klick auf *Return* verschwindet. Um die ganzen Textfeld-Antwortgeber zu quittieren, wird die Methode *resignFirstResponder()* in *touchesBegan(...)* aufgerufen. Die zwei Funktionen *keyboardWillShow* und *keyboardWillHide* sind für die Verschiebung der Ansicht verantwortlich.

* + 1. Reflecton/ViewControllers/HomeViewController.swift

Das Herzstück unserer iOS-App stellt der *HomeViewController* dar. Dieser wurde zum Verschieben der Widgets, sowie zum Hinzufügen von E-Mail- und Kalender-Konten erstellt.



Abbildung 62: Widgets zum verschieben

Die vier abgebildeten Widgets können mithilfe von Halten und Drag&Drop verschoben werden. Dabei wird die Widget-Anordnung in die Datenbank gespeichert. Außerdem können noch E-Mail- bzw. Kalender-Konten hinzugefügt werden. Dies geschieht mit einem Klick auf das E-Mail- bzw. Kalender-Widget. Des Weiteren befindet sich in der linken unteren Ecke ein Zahnrad. Dieser Knopf öffnet den *WiFiViewController*.

Membervariablen

Jene Membervariablen, welche in der HomeViewController-Klasse benötigt werden, sind in der unteren Tabelle dargestellt.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| collectionView | Stellt eine Sammelansicht von verschiedenen Zellen dar. |
| widgets | Speichert mithilfe des Indexes die Widget-Anordnung. |
| longPressGesture | Wird das gewünschte Widget lange gedrückt, so wird dieses beweglich. |
| email | Hier ist die E-Mail-Adresse gespeichert |

Funktionen

Die unten gelisteten Funktionen sind notwendige Bestandteile der Hauptansicht *HomeViewController*.

Swift generiert standardmäßig eine Funktion namens *viewDidLoad()*. Jene wird aufgerufen, sobald der komplette *ViewController* in den Memory-Speicher geladen wurde.

|  |
| --- |
| **override func** viewDidLoad() {  **super**.viewDidLoad()  **self**.collectionView.backgroundColor = UIColor.clear   longPressGesture = UILongPressGestureRecognizer(target: **self**,  action: **#selector**(HomeViewController.handleLongGesture(\_:)))  **self**.collectionView.addGestureRecognizer(longPressGesture)   **self**.loadUserData() } |

Codeabschnitt 160: ViewController in Memory geladen (HomeViewController.swift)

Die *UICollectionView* hat einen durchsichtigen Hintergrund und besteht aus verschiedenen Widgets bzw. Zellen. Diese Zellen sollen, wie gesagt, durch langes Halten beweglich sein und benötigen daher einen *UILongPressGestureRecognizer*. Da die Funktion *viewDidLoad()* die erste Funktion ist, welche automatisch aufgerufen wird, ist es sinnvoll die Daten mithilfe von *loadUserData()* herunterzuladen. Die von Swift selbsterstellte Funktion *didReceiveMemoryWarning()* wird aufgerufen, wenn ein Problem mit dem Memory-Speicher vorliegt.

|  |
| --- |
| **override func** didReceiveMemoryWarning() {  **super**.didReceiveMemoryWarning() } |

Codeabschnitt 161: Memory-Problemwarnung (HomeViewController.swift)

Innerhalb des Methodenblocks wird die Funktion *didReceiveMemoryWarning()* der Superklasse *ViewController* aufgerufen.

Im *HomeViewController* befindet sich genau ein Knopf. Der sogenannte *settingsButton*. Mit ihm gelangt der Benutzer auf die *WiFiViewController*-Ansicht, in jener das Netzwerk des Spiegels konfigurierbar ist.

|  |
| --- |
| **@IBAction func** settingsButton(\_ sender: UIButton) {  **self**.performSegue(withIdentifier: **"SegueToRight"**, sender: sender) } |

Codeabschnitt 162: Aktion Settings-Button (HomeViewController.swift)

Wird der Zahnrad-Knopf gedrückt, wird ein Übergang nach rechts ausgeführt. Beim Übergang vom *HomeViewController* auf den *WiFiViewController* ist allerdings eine Vorbereitung mithilfe der *prepare(...)* Funktion notwendig. Dabei darf aber auch auf das Übergeben der E-Mail nicht vergessen werden, da diese im *WiFiViewController* benötigt wird. Zum einen wird die E-Mail an das PHP-Skript des Raspberry Pis übergeben. Dieses speichert dann die Mail-Adresse in einem lokalen File. Zum anderen wird sie benötigt, wenn der Benutzer vom *WiFiViewController* wieder auf den *HomeViewController* zurück wechselt.

|  |
| --- |
| **override func** prepare(for segue: UIStoryboardSegue, sender: **Any**?) {  **let** nextViewController = segue.destination **as**! WiFiViewController  nextViewController.email = **self**.email } |

Codeabschnitt 163: Vorbereitung des Überganges (HomeViewController.swift)

Der Ziel-*ViewController* ist im *HomeViewController* auf jeden Fall der *WiFiViewController*. Wie bereits oben erwähnt, wird auch die E-Mail-Adresse übergeben. Dabei wird einfach die Variable *email* des *WiFiViewControllers* gesetzt.

Innerhalb von *viewDidLoad()* wurde die Funktion *handleLongGesture(...)* aufgerufen. Diese kümmert sich um die Abhandlung der Geste des Verschiebens.

|  |
| --- |
| **@objc func** handleLongGesture(\_ gesture: UILongPressGestureRecognizer) {  **switch**(gesture.state) {  **case** UIGestureRecognizerState.began:  **guard let** selectedIndexPath = **self**.collectionView.indexPathForItem(  at: gesture.location(in: **self**.collectionView))  **else** {  **break** }  collectionView.beginInteractiveMovementForItem(at: selectedIndexPath)   **case** UIGestureRecognizerState.changed:  collectionView.updateInteractiveMovementTargetPosition(gesture.location(in:   gesture.view!))   **case** UIGestureRecognizerState.ended:  collectionView.endInteractiveMovement()   **default**:  collectionView.cancelInteractiveMovement()  } } |

Codeabschnitt 164: Drag&Drop-Geste (HomeViewController.swift)

Wird ein Widget bzw. eine Zelle gehalten und bewegt, so wird vorher mit der Funktion *beginInteractiveMovementForItem(...)* eine Bewegung begonnen. Wird das Widget an einen anderen Ort bewegt, aber nicht fallen gelassen, so wird im zweiten *case*-Statement die Zielposition in das *CollectionView*-Objekt eingetragen. Lässt der Benutzer das Widget am gewünschten Zielort wieder aus bzw. fallen, so wird im dritten *case*-Statement die Bewegung beendet. Es gibt allerdings auch noch den *default*-Fall, bei jenem die Bewegung einfach abgebrochen wird.

Im *HomeViewController* befinden sich drei Funktionen, die auf den Namen *collectionView(...)* hören. Diese werden bei bestimmten Tätigkeiten des Benutzers aufgerufen.

|  |
| --- |
| *// 1. Function* **func** collectionView(\_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  **return** widgets.count }  *// 2. Function* **func** collectionView(\_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {  **let** cell = collectionView.dequeueReusableCell(withReuseIdentifier: **"Cell"**,  for: indexPath) **as**! TextCollectionViewCell  **var** image: UIImage = UIImage()  cell.textLabel.text = **""   if** widgets[indexPath.item].range(of: **"email"**) != **nil** {  image = UIImage(named: **"WidgetEmail"**)!  }  **else if** widgets[indexPath.item].range(of: **"calendar"**) != **nil** {  image = UIImage(named: **"WidgetCalendar"**)!  }  **else if** widgets[indexPath.item].range(of: **"clock"**) != **nil** {  image = UIImage(named: **"WidgetClock"**)!  }  **else if** widgets[indexPath.item].range(of: **"weather"**) != **nil** {  image = UIImage(named: **"WidgetWeather"**)!  }  **else** {  cell.textLabel.text = widgets[indexPath.item]  }   **let** imageView = UIImageView(image: image)  cell.backgroundView = UIView()  cell.backgroundView!.addSubview(imageView)  cell.addGestureRecognizer(UITapGestureRecognizer(target: **self**,  action: **#selector**(tap(\_:))))   **return** cell }  *// 3. Function* **func** collectionView(\_ collectionView: UICollectionView, moveItemAt sourceIndexPath: IndexPath, to destinationIndexPath: IndexPath) {  **let** temp = widgets.remove(at: sourceIndexPath.item)  widgets.insert(temp, at: destinationIndexPath.item)  **var** data = **""   for** i **in** 0..<widgets.count {  data = data + **"<widget>"** + widgets[i] + **"</widget>"** }  \_ = **self**.setUserData(data: data) } |

Codeabschnitt 165: Funktionen der CollectionView (HomeViewController.swift)

Bei der ersten Funktion wird die Anzahl der Widgets zurückgegeben. Sie wird von der *CollectionView* automatisch aufgerufen und gibt an, wie viele Zellen generiert werden sollen.

Auch die zweite Funktion wird von der *CollectionView* automatisch ausgeführt. Die Methode spezialisiert jede einzelne Zelle unter Verwendung des Indexes. Spezialisieren im Sinne von Festlegen des Hintergrundbildes, sowie Hinzufügen der Gestenerkennung.

Im Gegensatz zu den anderen, stellt die dritte Funktion den maßgeblichen Teil zur Funktionsfähigkeit bereit. Wenn der Benutzer ein Widget bzw. eine Zelle bewegt, verändert sich die Anordnung von diesem. Es wird also das Widget am Anfangsort gelöscht und am Zielort hinzugefügt. Dafür sind die Funktionen *remove(...)* bzw. *insert(...)* verantwortlich. Die neue Anordnung wird schlussendlich in der *data*-Variable gespeichert, welche in die Datenbank mithilfe von *setUserData(...)* geladen wird.

Die Bewegung der Widgets ist zwar recht schön anzusehen, aber dies reicht noch nicht aus. Die Widgets E-Mail und Kalender müssen, wie gesagt, interaktiv gestaltet sein. Das heißt, dass mit einem Klick auf eines der beiden, sich ein Anmeldefenster für einen Mail- bzw. Kalender-Account öffnen muss. Dies wird durch die Funktion *tap(...)* verwirklicht.

|  |
| --- |
| **@objc func** tap(\_ sender: UITapGestureRecognizer) {  **let** location = sender.location(in: **self**.collectionView)  **let** indexPath = **self**.collectionView.indexPathForItem(at: location)   **if** widgets[(indexPath?.item)!].range(of: **"email"**) != **nil** {  **let** alertController = UIAlertController(title: **"Add E-Mail Account"**,  message: **"Enter your E-Mail and Password"**,   preferredStyle: UIAlertControllerStyle.alert)  alertController.addTextField(configurationHandler: { (textField) **in** textField.placeholder = **"E-Mail"** textField.keyboardType = .emailAddress  })  alertController.addTextField(configurationHandler: { (textField) **in** textField.placeholder = **"Password"** textField.isSecureTextEntry = **true** })  alertController.addAction(UIAlertAction(title: **"OK"**, style: .default, handler: {  (action) **in  let** textEmail = alertController.textFields![0] **as** UITextField  **let** textPassword = alertController.textFields![1] **as** UITextField  **var** data = **""   for** i **in** 0..<**self**.widgets.count {  **if self**.widgets[i].range(of: **"email"**) != **nil** {  data = data + **"<widget>"** + **self**.widgets[i] + **":"** + textEmail.text!  + **":"** + textPassword.text! + **"</widget>"** }  **else** {  data = data + **"<widget>"** + **self**.widgets[i] + **"</widget>"** }  }  **self**.saveUserData(data: data)  \_ = **self**.setUserData(data: data)  }))  alertController.addAction(UIAlertAction(title: **"Cancel"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  }  **else if** widgets[(indexPath?.item)!].range(of: **"calendar"**) != **nil** {  **let** alertController = UIAlertController(title: **"Add Calendar"**,  message: **"Enter your E-Mail and Password"**,  preferredStyle: UIAlertControllerStyle.alert)   alertController.addTextField(configurationHandler: { (textField) **in** textField.placeholder = **"E-Mail"** textField.keyboardType = .emailAddress  })  alertController.addTextField(configurationHandler: { (textField) **in** textField.placeholder = **"Password"** textField.isSecureTextEntry = **true** })  alertController.addAction(UIAlertAction(title: **"OK"**, style: .default, handler: {  (action) **in  let** textEmail = alertController.textFields![0] **as** UITextField  **let** textPassword = alertController.textFields![1] **as** UITextField  **var** data = **""   for** i **in** 0..<**self**.widgets.count {  **if self**.widgets[i].range(of: **"calendar"**) != **nil** {  data = data + **"<widget>"** + **self**.widgets[i] + **":"** + textEmail.text!  + **":"** + textPassword.text! + **"</widget>"** }  **else** {  data = data + **"<widget>"** + **self**.widgets[i] + **"</widget>"** }  }  **self**.saveUserData(data: data)  \_ = **self**.setUserData(data: data)  }))  alertController.addAction(UIAlertAction(title: **"Cancel"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  } } |

Codeabschnitt 166: Pop-Up-Fenster für E-Mail- bzw. Kalender-Account (HomeViewController.swift)

In der äußeren *if*-Anweisung wird zwischen dem E-Mail-Widget und dem Kalender-Widget unterschieden. Der Inhalt der beiden Blöcke ist aber fast ebengleich. Unabhängig vom Widget wird ein *UIAlertController* erstellt, welcher Textfelder für E-Mail und Passwort besitzt, sowie zwei Knöpfe bietet.

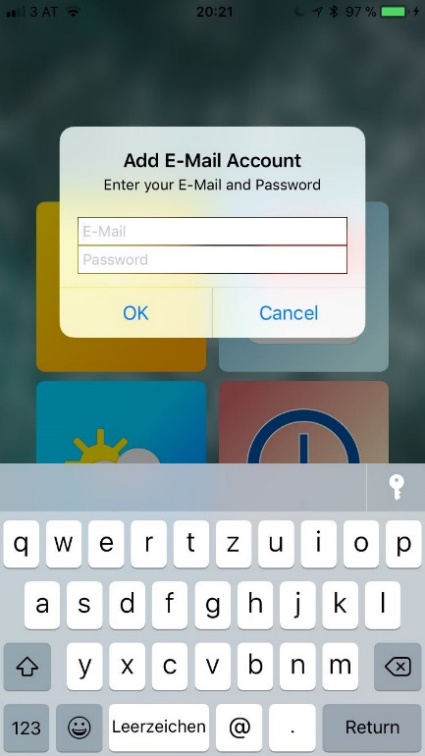


Abbildung 63: Konto hinzufügen

Nachdem der Benutzer seine Daten nach der Eingabe bestätigt hat, werden diese in der Datenbank mithilfe des Trennzeichens „:“ gespeichert. Für diesen Vorgang wird die Funktion *setUserData(...)* verwendet. Bei hinzugefügten E-Mail- und Kalender-Account sieht das *data*-Objekt in der Datenbank dann wie folgt aus:

|  |
| --- |
| <widget>email**:m.mustermann@mail.com:pw1**</widget>  <widget>calendar**:m.mustermann@icloud.com:pw2**</widget>  <widget>weather</widget>  <widget>clock</widget>" |

Codeabschnitt 167: Auf Account zugeschnittene Widgets

Wie oben ersichtlich sind innerhalb der Widgets E-Mail und Kalender die Mail-Adresse bzw. das zugehörige Passwort des jeweiligen Accounts gespeichert. Jene Daten werden vom Java-UI-Programm heruntergeladen. Dadurch sieht der Benutzer dann alle beliebigen E-Mails und Termine am Spiegel.

In unserem *HomeViewControllers* findet ein ständiger Datenaustausch mit dem Datenbank-Server statt. Mithilfe der Funktionen *loadUserData()* und der von *loadUserData()* aufgerufenen Funktion *getUserData(...)* werden Daten vom Server heruntergeladen.

|  |
| --- |
| **func** loadUserData() {  **self**.getUserData { jsonString **in  let** widgetsArrangement = String(jsonString)  **var** widgetsArrangementString = widgetsArrangement.replacingOccurrences(  of: **"</widget>"**, with: **"/"**).replacingOccurrences(of: **"<widget>"**, with: **""**)  widgetsArrangementString.removeLast()  **let** widgetsArrangementArray = widgetsArrangementString.components(  separatedBy: **"/"**)  **self**.widgets.removeAll()   **for** i **in** 0..<widgetsArrangementArray.count {  **self**.widgets.append(widgetsArrangementArray[i])  }  DispatchQueue.main.async {  **self**.collectionView.reloadData()  }  } }  **func** getUserData(completion: **@escaping** (String) -> ()) {  **let** urlInterface = URL(string:   **"http://**\(ipAddressServer)**/Reflecton/api/getdata.php"**)  **if** !**self**.email.isEmpty {  **let** postParameters = **"email="** + **self**.email   **var** request = URLRequest(url: urlInterface!)  request.httpMethod = **"POST"** request.httpBody = postParameters.data(using: .utf8)   **let** task = URLSession.shared.dataTask(with: request) {  data, response, error **in  if let** data = data, **let** jsonString = String(data: data,   encoding: String.Encoding.utf8), error == **nil** {  completion(jsonString)  }  **else** {  print(**"error=**\(error!.localizedDescription)**"**)  }  }  task.resume()  } } |

Codeabschnitt 168: Laden der Benutzerdaten (HomeViewController.swift)

Innerhalb der Funktion *getUserData(...)* wird das *data*-Feld des jeweiligen Reflecton-Accounts mithilfe des PHP-Skripts heruntergeladen. Das E-Mail-Feld kommt ursprünglich vom *SignInViewController* und dient innerhalb der Datenbank als Schlüssel. Die Funktion *getUserData(...)* wird von *loadUserData(...)* aufgerufen. Die Methode *loadUserData(...)* wandelt den ursprünglich vom Server erhaltenen *jsonString* in eine Zwischenvariable *widgetsArrangementArray* um. Mithilfe von dieser wird die Membervariable *widgets* mit den aktuellen Werten bzw. der aktuellen Anordnung über den Index ersetzt. Nach dem Aufruf der von Swift bereitgestellten Funktion *reloadData()* wird die aktuelle Anordnung des *widgets*-Arrays in die *CollectionView* übernommen.

Neben den komplexeren Datenbank-Funktionen gibt es noch eine Methode namens *saveUserData(...)* die zur lokalen Speicherung der Widget-Anordnung bzw. der *data*-Variable verantwortlich ist.

|  |
| --- |
| **func** saveUserData(data: String) {  **var** widgetsArrangementString = data.replacingOccurrences(of: **"</widget>"**,  with: **"/"**).replacingOccurrences(of: **"<widget>"**, with: **""**)  widgetsArrangementString.removeLast()   **let** widgetsArrangementArray = widgetsArrangementString.components(separatedBy: **"/"**)  **self**.widgets.removeAll()   **for** i **in** 0..<widgetsArrangementArray.count {  **self**.widgets.append(widgetsArrangementArray[i])  } } |

Codeabschnitt 169: Lokales Speichern der Widget-Anordnung (HomeViewController.swift)

Beim Hinzufügen eines E-Mail- bzw. Kalender-Accounts wird obige Funktion benötigt. Dabei wird das *data*-Feld zuerst in eine lokale Zwischenvariable namens *widgetsArrangementArray* umgewandelt. Mithilfe von dieser wird die Membervariable *widgets* mit den aktuellen Werten bzw. der aktuellen Anordnung über den Index ersetzt.

Die Funktion *setUserData(...)* ist zum Speichern einer als Parameter übergebenen *data*-Variable verantwortlich. Als Schlüssel wird wie üblich die E-Mail-Adresse des Reflecton-Accounts verwendet. Diese Funktion wird aufgerufen, wenn sich die Anordnung der Widgets geändert hat, oder ein E-Mail- bzw. Kalender-Account hinzugefügt wurde.

|  |
| --- |
| **func** setUserData(data: String) -> Int {  **let** urlInterface = URL(string:   **"http://**\(ipAddressServer)**/Reflecton/api/setdata.php"**)   **if** !**self**.email.isEmpty && !data.isEmpty {  **let** postParameters = **"email="** + **self**.email + **"&data="** + data   **var** request = URLRequest(url: urlInterface!)  request.httpMethod = **"POST"** request.httpBody = postParameters.data(using: .utf8)   **let** task = URLSession.shared.dataTask(with: request) { data, response, error **in guard let** data = data, error == **nil else** {  print(**"error=**\(String(describing: error))**"**)  **return** }  **if let** httpStatus = response **as**? HTTPURLResponse, httpStatus.statusCode   != 200 {  print(**"statusCode should be 200, but is** \(httpStatus.statusCode)**"**)  print(**"response =** \(String(describing: response))**"**)  }  \_ = String(data: data, encoding: .utf8)  }  task.resume()  **return** 1  }  **else** {  **return** 0  } } |

Codeabschnitt 170: Aktualisierung der Daten (HomeViewController.swift)

Um die *data*-Variable in der Datenbank im *data* -Feld speichern zu können, wird ein PHP-Skript aufgerufen, welches die Schnittstelle zwischen iOS-App und Datenbank darstellt. Dieses Skript gibt im Regelfall eine 1 zurück.

* + 1. Reflecton/ViewControllers/WiFiViewController.swift

Zu guter Letzt folgt noch der *WiFiViewController*. Damit dieser ordnungsgemäß arbeiten kann, muss das iPhone mit dem Hotspot des Raspberrys verbunden sein. Dies geschieht ganz einfach über die Systemeinstellungen von iOS. Der Hotspot wird, wie in einem anderen Kapitel bereits genauer beschrieben, ausgestrahlt, wenn der Pi nicht mit einem Netzwerk verbunden ist. In diesem *ViewController* werden bei eingegebener SSID und Passkey die Parameter *email*, *ssid* und *psk* an das PHP-Interface am Pi übertragen. Jenes PHP-Interface läuft am Apache Server des Pis unter der lokalen Adresse *192.168.50.5*.

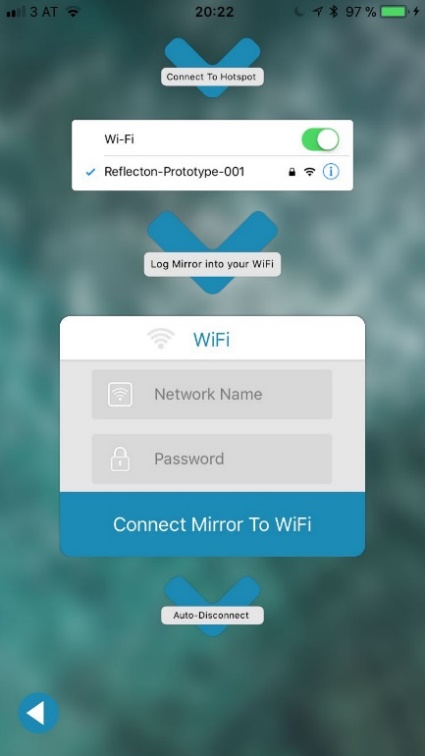


Abbildung 64: Konfiguration des WLAN-Netzwerkes

Unter anderem speichert das oben genannte PHP-Skript den Parameter *email* in eine bestimmte Textdatei, sodass das Java-UI-Programm den Parameter an der Datenbank als Schlüssel verwenden kann. Damit dies auch möglich ist, wurde in der *prepare(...)*-Funktion der eben besprochenen Klasse *HomeViewController* die Membervariable *email* des *WiFiViewControllers* gesetzt. Die Variable *email* wurde sozusagen übergeben.

Membervariablen

Im *WiFiViewController* gibt es diverse unten angeführte Membervariablen.

|  |  |
| --- | --- |
| **Membervariable** | **Bedeutung** |
| textfieldSSID | In diesem Textfeld ist die SSID des WiFi-Netzwerks einzutragen. |
| textfieldPSK | Zur SSID gehört natürlich auch ein Passwort des WLAN-Netzwerkes. |
| email | Hier ist die E-Mail-Adresse gespeichert |

Funktionen

Nun werden im Fließtext wieder alle möglichen Funktionen der Klasse dargestellt.

Die Funktion *viewDidLoad()* wird aufgerufen, sobald der komplette *ViewController* in den Memory-Speicher geladen wurde

|  |
| --- |
| **override func** viewDidLoad() {  **super**.viewDidLoad()  textfieldSSID.delegate = **self** textfieldPSK.delegate = **self** NotificationCenter.default.addObserver(**self**,  selector: **#selector**(SignUpViewController.keyboardWillShow(sender:)),  name: NSNotification.Name.UIKeyboardWillShow, object: **nil**)   NotificationCenter.default.addObserver(**self**,  selector: **#selector**(SignUpViewController.keyboardWillHide(sender:)),  name: NSNotification.Name.UIKeyboardWillHide, object: **nil**) } |

Codeabschnitt 171: ViewController in Memory geladen (WiFiViewController.swift)

Das Delegat (engl. delegate) muss bei jedem einzelnen Textfeld gesetzt werden. Somit weiß das Textfeld über den zugehörigen *ViewController* Bescheid. Die zwei weiteren Funktionsaufrufe mit *addObserver(...)* werden zum Zeigen bzw. Verstecken der virtuellen Tastatur benötigt.

Die von Swift selbsterstellte Funktion *didReceiveMemoryWarning()* wird aufgerufen, wenn ein Problem mit dem Memory-Speicher vorliegt.

|  |
| --- |
| **override func** didReceiveMemoryWarning() {  **super**.didReceiveMemoryWarning() } |

Codeabschnitt 172: Memory-Problemwarnung (WiFiViewController.swift)

Innerhalb des Methodenblocks wird die Funktion *didReceiveMemoryWarning()* der Superklasse *ViewController* aufgerufen.

Die Deklaration der unten abgebildeten Funktion *buttonConnect(...)* wird automatisch ausgeführt, sobald der *Connect*-Button mit gedrückter *CTRL*-Taste vom Storyboard in die Klasse *WiFiViewController* gezogen wird.

|  |
| --- |
| **@IBAction func** buttonConnect(\_ sender: UIButton) {  **let** urlInterface = URL(string: **"http://**\(ipAddressMirror)**/ReflectonSetup/setup.php"**)  **let** ssid = textfieldSSID.text  **let** psk = textfieldPSK.text   **if** !ssid!.isEmpty && !psk!.isEmpty {  **let** postParameters = **"email="** + **self**.email + **"&ssid="** + ssid! + **"&psk="** + psk!  **var** request = URLRequest(url: urlInterface!)  request.httpMethod = **"POST"** request.httpBody = postParameters.data(using: .utf8)   **let** task = URLSession.shared.dataTask(with: request) { data, response, error **in  guard let** data = data, error == **nil else** {  print(**"error=**\(String(describing: error))**"**)  **return** }  **if let** httpStatus = response **as**? HTTPURLResponse, httpStatus.statusCode !=   200 {  print(**"statusCode should be 200, but is** \(httpStatus.statusCode)**"**)  print(**"response =** \(String(describing: response))**"**)  }  \_ = String(data: data, encoding: .utf8)  }  task.resume()  }  **else** {  **let** alertController = UIAlertController(title: **"Reflecton WiFi Controller"**,   message: **"Please check your entries!"**,  preferredStyle: UIAlertControllerStyle.alert)  alertController.addAction(UIAlertAction(title: **"OK"**,  style: UIAlertActionStyle.default,handler: **nil**))  **self**.present(alertController, animated: **true**, completion: **nil**)  } } |

Codeabschnitt 173: Aktion Connect-Button (WiFiViewController.swift)

Die Variablen *email*, *ssid* und *psk* werden an ein PHP-Skript übertragen, welches auf dem Raspberry Pi läuft. Dieses PHP-Interface speichert die *email*-Variable in einer bestimmten Textdatei und konfiguriert die *wpa\_supplicant.conf* hinsichtlich der neuen *ssid* und *psk*.

Neben dem *Connect*-Button gibt es auch noch den *Back*-Button, mit welchem man auf den *HomeViewController* zurückkommt.

|  |
| --- |
| **@IBAction func** backButton(\_ sender: UIButton) {  **self**.performSegue(withIdentifier: **"SegueToLeft"**, sender: sender) } |

Codeabschnitt 174: Aktion Back-Button (WiFiViewController.swift)

Innerhalb der Funktion wird einfach nur ein Übergang mithilfe der Klasse *SegueToLeft* ausgeführt.

Beim Übergang vom *WiFiViewController* zurück zum *HomeViewController* ist allerdings eine Vorbereitung mithilfe der prepare(...) Funktion notwendig. Dabei darf aber auch auf das Übergeben der E-Mail nicht vergessen werden, da diese im *HomeViewController* wieder benötigt wird.

|  |
| --- |
| **override func** prepare(for segue: UIStoryboardSegue, sender: **Any**?) {  **let** nextViewController = segue.destination **as**! HomeViewController  nextViewController.email = **self**.email } |

Codeabschnitt 175: Vorbereitung für Übergang (WiFiViewController.swift)

Der Ziel-*ViewController* ist im *WiFiViewController* auf jeden Fall der *HomeViewController*. Wie bereits oben erwähnt, wird auch die E-Mail-Adresse übergeben. Dabei wird einfach die Variable *email* des *HomeViewControllers* gesetzt.

Bei einem Klick auf ein Textfeld soll die virtuelle Tastatur von iOS angezeigt werden. Klickt der Benutzer dann auf eine andere Stelle als auf die Tastatur oder Textfeld, dann soll die virtuelle Tastatur wieder ausgeblendet werden.

|  |
| --- |
| **func** textFieldShouldReturn(\_ textField: UITextField) -> Bool {  textField.resignFirstResponder()  **return true** }  **override func** touchesBegan(\_: Set<UITouch>, with: UIEvent?) {  textfieldSSID.resignFirstResponder()  textfieldPSK.resignFirstResponder()  **self**.view.endEditing(**true**) }  **@objc func** keyboardWillShow(sender: NSNotification) {  **self**.view.frame.origin.y = -160 }  **@objc func** keyboardWillHide(sender: NSNotification) {  **self**.view.frame.origin.y = 0 } |

Codeabschnitt 176: Anzeigen der virtuellen Tastatur (WiFiViewController.swift)

Die Funktion *textFieldShouldReturn(...)* sorgt dafür, dass die virtuelle Tastatur bei einem Klick auf *Return* verschwindet. Um die ganzen Textfeld-Antwortgeber zu quittieren, wird die Methode *resignFirstResponder()* in *touchesBegan(...)* aufgerufen. Die zwei Funktionen *keyboardWillShow* und *keyboardWillHide* sind für die Verschiebung der Ansicht verantwortlich.

1. **Android-App**

Nachdem die iOS-App bereits erfolgreich abgeschlossen wurde, kam die Idee auf, dieses Konzept auch für Android-Nutzer zur Verfügung zu stellen. Diese App sollte es also ebenfalls dem Benutzer ermöglichen, den Spiegel auf eine angenehme Art und Weise zu konfigurieren und personalisieren. Da die Zeit jedoch schon knapp war, stand von Anfang an ein Fragezeichen über der Fertigstellung dieses Projekts. Dennoch oder gerade deswegen wurde von Anfang an akribisch an der Entwicklung dieser App gearbeitet.

* 1. **Vorbereitung**

Vor dem Beginn der Programmierung mussten einige Dinge geklärt werden. Unter anderem gehört im Vorhinein eine angedachte Programmstruktur auf Papier gebracht, um schnell einen guten Überblick über das Gesamtprodukt zu erlangen. Auch muss über gewisse von Android verlangte Grundbedingungen wie eine Minimum-SDK nachgedacht werden. Den Beginn stellte allerdings die Wahl einer Entwicklungsumgebung, mit der eine Android-App von diesem Typ und Umfang bestmöglich erstellt und auf einem passenden Emulator vorab getestet werden kann.

12.1.1 Erstellungsdetails

Die beiden im Android-Sektor am verbreitetsten Entwicklungsumgebungen sind Android Studio und IntelliJ. Es wurde schlussendlich Android Studio gewählt, da hiermit schon im Zuge des vorjährigen HWE-Projekts Erfahrungen gesammelt wurden und Android Studio sowieso auf IntelliJ basiert.

Beim Erstellen des Projekts musste eine Minimum-Android-SDK-Version gewählt werden. Hierbei fiel die Wahl auf Version 5.1 (Lollipop), welche im Wesentlichen alle benötigten Funktionen beinhaltete und zum Erstellungszeitpunkt bei 63% aller Android-Nutzer bereits installiert war.

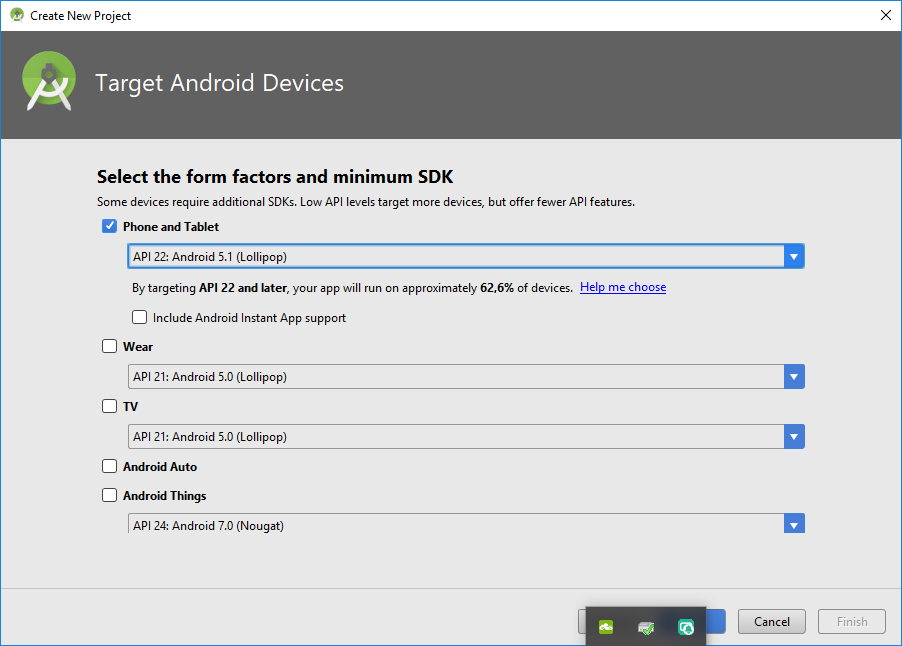


Abbildung 65: Wählen der SDK

Danach konnte das Projekt erstellt und mit dem Ausarbeiten der Programmstruktur begonnen werden. Hierbei wurde festgelegt, in der *MainActivity* die Start-Anwendung zu kreieren, in welcher dann der Endnutzer sich mit seinem Profil einloggen kann. Dazu wurden Java-Klassen für die Registrierungs-Aktivität, die Widgets-Anordnungs-Aktivität, die Einstellungen-Aktivität sowie zur Datenbankverbindung und Account-Bearbeitung ausprogrammiert.

* + 1. Design

Designtechnisch war von Anfang an die iOS-App das Vorbild. Die Struktur soll für den Benutzer in keiner Weise unterschiedlich sein. Da jedoch die in der iOS-App dargestellten Bilder bei der Programmierung nicht verfügbar waren, wurden provisorische Beispielbilder verwendet, welche nach der Fertigstellung durch die Original-PNG-Dateien ersetzt wurden. Diese Bild-Dateien wurden im Projekt-Ordner unter *res* (resources) gespeichert, damit das Programm diese abgreifen kann.

* 1. **Programmierung**

Da alle im *values*-Ordner erstellten XML-Dateien bereits standardmäßig den Vorstellungen entsprachen, konnte die Programmierung damit begonnen werden, die Aktivitäten in der Design-Ansicht zu konfigurieren.

* + 1. Start-Anwendung

Dabei wurde mit der *activity\_main.xml* begonnen. In dieser wurde die Log In-Funktion dargestellt.

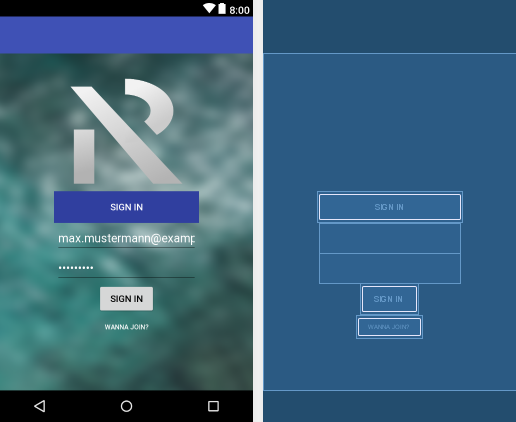


Abbildung 66: Main\_Activity

Dabei kamen unter einem titelgebenden Textfeld die Eingabefelder für die E-Mail-Adresse bzw. das persönliche Passwort, ein Button zur Durchführung des Log In-Prozesses und ein unscheinbarer Button zum Anlegen eines neuen Kontos. Drücken auf den „SIGN IN“-Knopf führt zuerst zur Überprüfung der Eingabe auf Einhalten eines bestimmten Schemas, welches unter anderem vorschreibt, dass die E-Mail-Adresse ein „@“ beinhalten muss oder das Passwort zumindest acht Zeilen lang sein muss. Wenn diese Vorgaben eingehalten sind, wird geprüft, ob sich die Inhalte mit einem Eintrag in der Datenbank decken. Sollte dies auch der Fall sein, wechselt die Ansicht zur aktuellen individuellen Widget-Ansicht. Wenn nicht, erscheint eine Pop-up-Nachricht, welche auf den Fehler in der Eingabe hinweist.

Durch den „WANNA JOIN?“-Knopf öffnet sich eine neue Anwendung, in welcher ein neues Profil angelegt werden kann.

In der zugehörigen Java-Klasse mussten zuallererst alle Bausteine der Ansicht eingelesen werden. Dies geschah über die folgende schematische Codezeile:

|  |
| --- |
| **public** Button join\_button = (Button) findViewById(R.id.Join\_Button); |

Codeabschnitt 177: Deklaration des „Wanna Join?“-Buttons (MainActivity.java)

Sobald dies geschehen war, konnte die auszuführende Aktion bei Betätigen der Buttons ausprogrammiert werden. Bei Betätigen des *SignIn\_Button* soll ein neues Fenster geöffnet werden, in welchem die Widgets individualisierbar sind.

|  |
| --- |
| sign\_in\_button.setOnClickListener(new View.OnClickListener(){  public void onClick(View view){  String[] dbString = dbHandler.databaseToString().split(" ");  for(int i = 0; i < 10000; i++) {  if (dbString[i].equals(email\_field.getText()) && dbString[i+1].equals(password\_field.getText())) {  Intent intent = new Intent(MainActivity.this, Widgets\_Activity.class);  startActivity(intent);  setContentView(R.layout.activity\_widgets\_);  }  }  } }); |

Codeabschnitt 178: Funktion für Aktion bei Click auf Button (MainActivity.java)

Dies ist jedoch an Bedingungen geknüpft. Die Eingaben müssen nämlich mit denen eines sich in der Datenbank befindlichen Account übereinstimmen. Diese Überprüfung findet in einer for-Schleife statt, welche alle Datenbankeinträge durchläuft und im Falle einer Übereinstimmung per *Intent* in die Widgets-Aktivität wechselt.

Dazu muss der Account, bei dem die Einigkeit gefunden wurde, aus der Liste aller Accounts ausgewählt werden, um die Darstellung so ordnen zu können, wie sie von diesem Nutzer zuletzt genutzt wurde.

|  |
| --- |
| Map<String, Account> newAccountMap = new HashMap<~>(); accountsInOrder = new LinkedList<Account>(); String accountUuids = getPreferences().get("AccountsUuid", null); if ((accountUuids != null) && (accountUuids.split() != 0)) {  String[] uuids = accountUuids.split(“,”);  for (String uuid : uuids) {  Account account = accounts.get(uuid);  if (account != null) {  newAccountMap.put(uuid, account);  accountsInOrder.add(account);  } else {  Account newAccount = new Account(null);  newAccountMap.put(uuid, newAccount);   accountsInOrder.add(account);  break;  }  } } |

Codeabschnitt 179: Senden der eingegebenen Accountdaten an Liste, Überprüfen auf Übereinstimmung (MainActivity.java)

Sollte der Button mit der Aufschrift „WANNA JOIN?“ betätigt werden, wird direkt in ein neues Fenster gewechselt, in welchem ein neues Konto angelegt werden kann.

Wird in die Eingabefelder geklickt, um die persönlichen Daten einzugeben, wird der standardmäßige Inhalt zuvor gänzlich entfernt.

* + 1. Datenbankverbindung

Um die benötigten Daten aller Accounts zu bekommen, welche für die Log In-Funktion ebenso wie zum Hinzufügen eines neuen Profils nach erfolgreicher Registrierung benötigt werden, muss eine Verbindung zur SQL-Datenbank hergestellt werden, welche auf einem Contabo-Server gespeichert ist. Um dies zu realisieren, müssen zuerst alle relevanten Daten der Datenbank in ein lokales String-Array eingelesen werden.

|  |
| --- |
| public DBHandler(Context context, String name, SQLiteDatabase.CursorFactory factory){  super(context, DATABASE\_NAME, factory, DATABASE\_VERSION); } |

Codeabschnitt 180: Erstellen des DBHandlers (DBHandler.java)

Hierzu gehören sowohl der Datenbankname als auch die Version der Datenbank.

Im Folgenden werden zum vollständigen Arbeiten mit der Datenbank auch Funktionen benötigt, welche es ermöglichen, die Datenbank zu einem String-Array umzuwandeln, einen Account hinzuzufügen und die Account-Liste zu aktualisieren.

In dieser Klasse wird außerdem definiert, was geschehen soll, wenn die Tabelle neu erstellt wird oder ein neuer Eintrag eingelegt werden soll.

|  |
| --- |
| @Override public void onCreate(SQLiteDatabase db){   String query = "CREATE TABLE " + TABLE\_ACCOUNTS + " ( " + COLUMN\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMN\_EMAIL + "TEXT, " + COLUMN\_FIRST\_NAME + "TEXT, " + COLUMN\_LAST\_NAME + "TEXT, " + COLUMN\_PASSWORD + "TEXT" + ")";  db.execSQL(query); }  @Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){ db.execSQL("DROP TABLE IF EXISTS " + TABLE\_ACCOUNTS);  onCreate(db); }  public void addAccount(Accounts accounts){  ContentValues values = new ContentValues();   values.put(COLUMN\_FIRST\_NAME, accounts.getFirst\_name());  values.put(COLUMN\_LAST\_NAME, accounts.getLast\_name());  values.put(COLUMN\_EMAIL, accounts.getEmail\_address());  values.put(COLUMN\_PASSWORD, accounts.getPassword());  SQLiteDatabase db = getWritableDatabase();  db.insert(TABLE\_ACCOUNTS, null, values); } |

Codeabschnitt 181: Standardfunktionen der Datenbank (DBHandler.java)

Um die Datenbank für das Java-Programm lesbar zu machen, war eine weitere Funktion nötig. Diese wandelt die Tabellen-Inhalte in ein String-Array um.

|  |
| --- |
| public String databaseToString(){  String dbString = "";  SQLiteDatabase db = getWritableDatabase();   String query = "SELECT \* FROM " + TABLE\_ACCOUNTS + " WHERE 1";   Cursor c = db.rawQuery(query, null);  c.moveToFirst();   while(!c.isAfterLast()){  if(c.getString(c.getColumnIndex("first\_name")) != null){  dbString += c.getString(c.getColumnIndex("first\_name"));  dbString += "\t";  }  c.moveToNext();  if(c.getString(c.getColumnIndex("last\_name")) != null){  dbString += c.getString(c.getColumnIndex("last\_name"));  dbString += "\t";  }  c.moveToNext();  if(c.getString(c.getColumnIndex("email\_address")) != null){  dbString += c.getString(c.getColumnIndex("email\_address"));  dbString += "\t";  }  c.moveToNext();  if(c.getString(c.getColumnIndex("password")) != null){  dbString += c.getString(c.getColumnIndex("password"));  dbString += "\n";  }  c.moveToNext();  }  db.close();  return dbString; } |

Codeabschnitt 182: Umwandlung der Datenbank in String-Array (DBHandler.java)

Weiters wurde eine Klasse benötigt, in welcher die allgemeine Struktur eines Accounts inklusive aller „getter“- und „setter“-Funktionen für einzelne Komponenten eines Accounts definiert ist.

* + 1. Registrierungsanwendung

Wie bereits erwähnt, wird nach Betätigen des *Join\_Button* ein neues Fenster geöffnet, in welchem man die Daten für ein neues Konto eingeben kann. Hierbei werden fünf Eingabefelder und zwei Buttons ersichtlich.

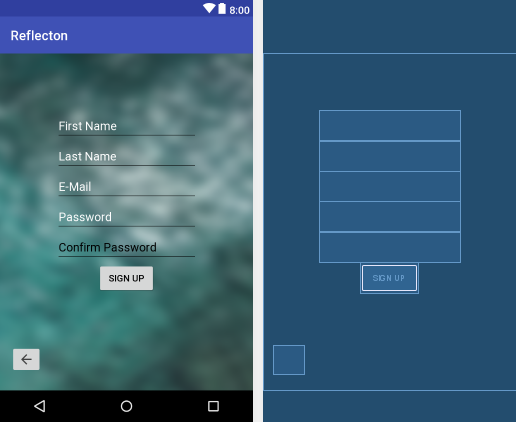


Abbildung 67: Sign\_Up\_Activity

Die fünf Eingabefelder beinhalten die Eingabe der Personendaten. Diese bestehen aus Vor- und Nachname, der E-Mail-Adresse und dem Passwort plus dessen Bestätigung. Wenn diese Felder korrekt ausgefüllt wurden und auf den *SignUp\_Button* gedrückt wurde, werden die eingegebenen Daten als neues Konto in der Datenbank gespeichert und man gelangt wie auch nach einem erfolgreichen Log In zur Widgets-Anwendung. Dabei werden die Daten auf diverse einzuhaltende Kriterien geprüft.

|  |
| --- |
| if (first\_name.length() >= 2 && last\_name.length() >= 2)) {  if (!first\_name.equals("First Name")) {  if (!last\_name.equals("Last Name")) {  if (email.contains("@") && email.contains(“.”) && email.length() >= 6){  if (password.equals(password\_confirmation)) {  Accounts accounts = new Accounts(first\_name, last\_name, email, password);  dbHandler.addAccount(accounts);  printDatabase();  } else  showAlert(view, "Password not confirmed correctly!");  } else  showAlert(view, "Incorrect e-mail-address");  } else  showAlert(view, "Last Name missing");  } else  showAlert(view, "First Name missing"); } else  showAlert(view, "Name too short"); } |

Codeabschnitt 183: Prüfen der Eingaben auf Richtigkeit (SignUpActivity.java)

Die Funktion *printDatabase()* sendet die bearbeitete Liste der Accounts schlussendlich an die Datenbank.

|  |
| --- |
| private void printDatabase() {  String dbFirstName = dbHandler.databaseToString();  first\_name\_input.setText(dbFirstName);  ... |

Codeabschnitt 184: Ausgeben der Datenbank-Einträge (SignUpActivity.java)

Zusätzlich befindet sich in dieser Anwendung in der linken unteren Ecke ein Button, welcher als Zurück-Pfeil dargestellt ist. Die Funktion dieses Buttons ist klar ersichtlich jene, aus der Registrierungs-Ansicht auszusteigen, um zurück zur Start-Ansicht zu geraten.

* + 1. Widgets-Anwendung

Wie bereits erwähnt, existieren mehrere Wege, um zur Widgets-Anwendung zu gelangen. Jedenfalls ist es nötig, seine persönlichen Daten anzugeben, um diese Ansicht sehen zu können.

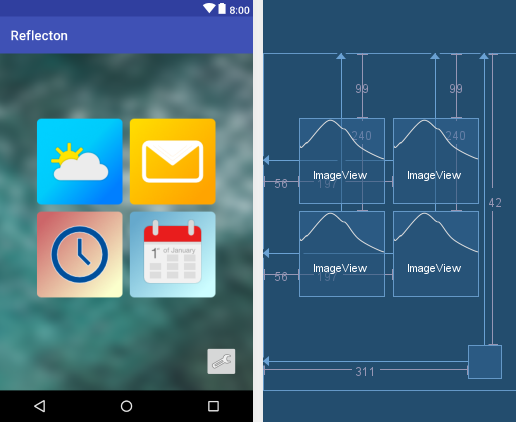


Abbildung 68: Widgets\_Activity

In dieser Ansicht befinden sich vier ImageViews rund um das Zentrum des Displays. Dazu kommt ein Button rechts unten mit einem Zangen-Symbol über welchen man zu den allgemeinen Einstellungen gelangt. Die ImageViews repräsentieren die vier Widgets, welche durch entsprechende Bild-Dateien erkenntlich dargestellt werden. Diese Widgets sind innerhalb der Default-Positionen wie in einer zweidimensionalen Array-Anordnung verschiebbar.

|  |
| --- |
| email\_view.setLayoutParams(layoutParams);  calendar\_view.setOnTouchListener(this); clock\_view.setOnTouchListener(this); weather\_view.setOnTouchListener(this); email\_view.setOnTouchListener(this); } public boolean onTouch(View view, MotionEvent event) {  final int X = (int) event.getRawX();  final int Y = (int) event.getRawY();  switch (event.getAction() & MotionEvent.ACTION\_MASK) {  case MotionEvent. ACTION\_DOWN:  RelativeLayout.LaoutParams lParams = (RelativeLayout.LayoutParams) view.getLayoutParams();  \_xDelta = X-lParams.leftMargin;   \_yDelta = Y-lParams.topMargin;  break;  case MotionEvent.ACTION\_MOVE:  RelativeLayout.LayoutParams layoutParams = (RelativeLayout.LayoutParams) view.getLayoutParams();  layoutParams.leftMargin = X - \_xDelta;   layoutParams.topMargin = Y - \_yDelta;   layoutParams.rightMargin = -250;   layoutParams.bottomMargin = -250;  view.setLayoutParams(layoutParams);  break;  }  email\_view.invalidate();  ...  return true; } |

Codeabschnitt 185: Funktion zum Verschieben der Widgets (WidgetsActivity.java)

Wird ein Widget an eine andere Position geschoben, wird das sich an dieser Position befindliche Gegenstück gleichzeitig an die vorherige Position dieses Widgets geschoben. So wird garantiert, dass niemals zwei Widgets an derselben Stelle positioniert sind.

* + 1. Einstellungen-Anwendung

Wie bereits erwähnt, öffnet sich bei Betätigen des in der Widget-Anwendung rechts unten positionierten Buttons ein neues Fenster, in welchem allgemeine Einstellungen ersichtlich werden. Genauer gesagt kann hier unter anderem mittels Switch die WiFi-Schnittstelle des Handys aktiviert bzw. deaktiviert werden. Weiters ist es möglich, Netzwerkdetails einzugeben und damit den Spiegel mit einer lokalen Internet-Quelle zu verbinden.

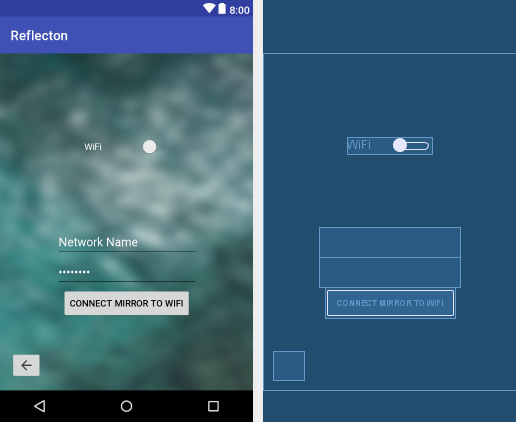


Abbildung 69: Settings\_Activity

Zusätzlich ist es auch hier wieder möglich, per „return“-Button aus dieser Ansicht zur Widgets-Anwendung zurückzukehren.

Aus zeitlichen Gründen konnte in dieser Anwendung einzig die Funktion des „return“-Buttons realisiert werden.

* 1. **Inbetriebnahme**

Die Applikation wurde regelmäßig mit einem Emulator getestet. Dazu musste im Vorfeld ein AVD (Android Virtual Device) erstellt werden. Dazu wurde mit dem Nexus 6 API 27 ein beispielhaftes Smartphone gewählt. Dies führte die Funktionen der App in ersten Feldversuchen einwandfrei aus.

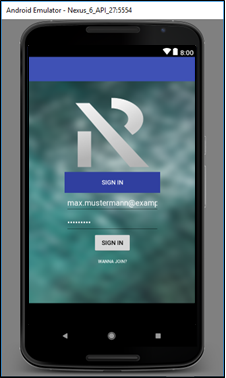


Abbildung 70: Start-Ansicht im Emulator

Nachdem die App um die versuchte interne Verbindung mit WLAN-Netzwerken erweitert wurde, konnte jedoch im Emulator nicht mehr auf Aktivitäten wie dem *onTouchListener* des *SignUp\_Button* zugegriffen werden. Das hieß, dass zwar immer noch in Textfelder oder auf Buttons gedrückt werden konnte, jedoch nicht mehr die Funktionen der diversen Listener, auch in der Start-Anwendung, ausgeführt wurden. Aus Zeitgründen wurde der Versuch der Beseitigung dieses Problems nach einigen Stunden abgebrochen und ein Rückschritt auf die vorhergehende Version war notwendig.

1. **Webseite**

Um die Funktionen nicht auf eine Android bzw. iOS-App zu beschränken, wurde als Erweiterung eine Website erstellt. Diese Website ermöglicht es dem Benutzer, dieselben Funktionen wie in der App nutzen zu können und einen groben Überblick über das Produkt „Reflecton“ zu bekommen.

* 1. **Startseite**

Auf der Startseite befinden sich allgemeine Informationen über das Projekt wie Anzahl der aktuellen Benutzer, Informationen über das Projektteam etc.

* + 1. Startseite in HTML

Zu allererst wird die Startseite in HTML programmiert. Dadurch werden die Platzverteilung und die Reihenfolge, in der die Informationen auf der Website angezeigt werden, festgelegt.

Um dieses Skript nicht ganz ohne Anhaltspunkte schreiben zu müssen, wurde eine Vorlage von *Bootstrap* verwendet.[[25]](#footnote-26) Diese Vorlage beinhaltete eine Platzaufteilung, Übergangseffekte und vorgefertigte Texte, welche übernommen oder geändert werden können.

In diesem Bereich der Website wird das Grundgerüst der Navigationsleiste festgelegt. Der Benutzer hat die Möglichkeit, durch einen Klick auf den betreffenden Link zu einer definierten Sektion auf der Startseite bzw. direkt zum Log In-Fenster der Website zu gelangen. Mithilfe eines *Containers* wird eine einheitliche Formatierung und Ausrichtung festgelegt, wodurch nicht jedes Element einzeln formatiert werden muss. Das *#*-Zeichen sorgt dafür, dass bestimmte Sektoren auf der Website erreicht werden können. Somit kann direkt ein bestimmter Bereich auf der Startseite oder eines anderen HTML-Dokumentes angezeigt werden.

|  |
| --- |
| <**header id="header"**>  <**div class="container"**>   <**div id="logo" class="pull-left"**>  <**h1**><**a href="#hero" style="text-align**: **left"**>Pinnovations</**a**></**h1**>  </**div**>   <**nav id="nav-menu-container"**>   <**ul class="nav-menu"**>  <**li class="menu-active"**><**a href="index.php#hero"**>Home</**a**></**li**>  <**li**><**a href="index.php#about"**>About Us</**a**></**li**>  <**li**><**a href="index.php#facts"**>Facts</**a**></**li**>  <**li**><**a href="index.php#services"**>Widgets</**a**></**li**>  <**li**><**a href="index.php#Frame"**>Frame</**a**></**li**>  <**li**><**a href="index.php#Sponsor"**>Sponsor</**a**></**li**>  <**li**><**a href="index.php#team"**>Team</**a**></**li**>  <**li**><**a href="index.php#call-to-action"**>Controll</**a**></**li**>  <**li**><**a href="login\_formular.php"**>Join us</**a**></**li**>  <**li**><**a href="index.php#contact"**>Contact Us</**a**></**li**>  </**ul**>  </**nav**>  </**div**> </**header**> |

Codeabschnitt 186: Header-Section in HTML

Für das Logo, welches am linken oberen Bereich der Navigationsleiste angezeigt wird, wurde mithilfe des Attributes *style* eine Textausrichtung nach links hinzugefügt. Dies hat zur Folge, dass dieses Element individuell formatiert werden kann ohne andere Elemente zu beeinflussen.

|  |
| --- |
| <**h1**><**a href="#hero" style="text-align**: **left"**>Pinnovations</**a**></**h1**> |

Codeabschnitt 187: Pinnovations Schriftzug in der Navigationsleiste

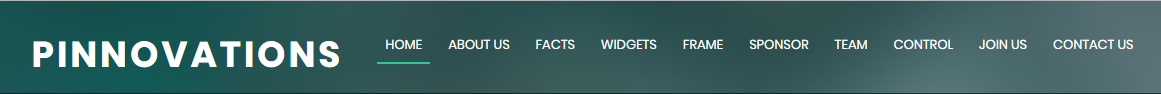


Abbildung 71: Navigationsleiste der Hauptseite

Durch zusätzliche Klassennamen wie *wow* und *fadInUp* werden die Effekte der jeweiligen Elemente gesteuert. Diese werden in CSS-Dateien beschrieben und ausprogrammiert.

|  |
| --- |
| <**div class="icon-box wow fadeInUp"**> |

Codeabschnitt 188: Übergangseffekte auf der Startseite

* + 1. Startseite in CSS

Da nun das Grundgerüst in HTML geschaffen ist, wird die Website mithilfe von CSS formatiert. Nun können die Elemente verschieden gestaltet und Effekte hinzugefügt werden.

HTML Elemente in CSS können durch den Namen ihrer Tags formatiert und angepasst werden. Es sind jede Menge verschiedener Stylings-Attribute verfügbar mit denen Schriftart, Farbe des Textes und Farbe des Hintergrundes eingestellt werden. Die Farbcodes können entweder hexadezimal oder mithilfe der *RGBA*-Tabelle angegeben werden.

Die Formatierung der reinen HTML-Tags ist in den meisten Fällen nicht ganz ausreichend. Daher können wie unterschiedlichen Bereiche, welche mit dem *div*-Tag ummantelt sind, einzeln formatiert werden.

|  |
| --- |
| **body** {  **background**: **#fff**;  **color**: **#666666**;  **font-family**: **"Open Sans"**, **sans-serif**; } |

Codeabschnitt 189: Formatierung Body-Tag in CSS

Mithilfe des *#-*Zeichen werden nun die selbstdefinierten Bereiche formatiert. Hierbei können ebenfalls wie bei den HTML spezifischen Tags verschiedene Attribute zugewiesen werden. Zusätzlich sind noch weitere Einstellungen wie Größe, Position etc. verfügbar.

|  |
| --- |
| **#header** {  **padding**: 30**px** 0;  **height**: 92**px**;  **position**: **fixed**;  **left**: 0;  **background**: **url**(**../img/hero-bg.png**) **top center fixed**;  **top**: 0;  **right**: 0;  **transition**: **all** 0.5**s**;  **z-index**: 997; } |

Codeabschnitt 190: Formatierung selbstdefinierter Kategorie in CSS

Um den HTML-Tag in einer benutzerdefinierten Kategorie individuell gestalten zu können, wird dieser nach dem Namen der Kategorie, welche mit einem # versehen wird, dazugeschrieben. Somit werden die Einstellungen nur an dem Tag in der Kategorie, und nicht wie sonst für alle auf der Seite vorhandenen Tags übernommen werden.

|  |
| --- |
| **#header #logo h1** {  **font-size**: 36**px**;  **margin**: 0;  **padding**: 6**px** 0;  **line-height**: 1;  **font-family**: **"Poppins"**, **sans-serif**;  **font-weight**: 700;  **letter-spacing**: 3**px**;  **text-transform**: **uppercase**; } |

Codeabschnitt 191: Angleichung der Formatierung für Logo und Navigationsleiste

Um den Button individuell gestalten zu können, wird dessen Name, welcher in HTML im sogenannten *class*-Tag definiert wird, mit einem Punkt davor nach dem Kategorienamen eingefügt. Somit werden die Einstellungen nur für den Button in der Kategorie verwendet und derselbe *class*-Name kann öfters verwendet werden. Dies gilt ebenso für den *Container-, Description-, Counters-*Tag o.ä.

|  |
| --- |
| **#hero** .**btn-get-started** {  **font-family**: **"Poppins"**, **sans-serif**;  **text-transform**: **uppercase**;  **font-weight**: 500;  **font-size**: 16**px**;  **letter-spacing**: 1**px**;  **display**: **inline-block**;  **padding**: 8**px** 28**px**;  **border-radius**: 50**px**;  **transition**: 0.5**s**;  **margin**: 10**px**;  **border**: 2**px solid #fff**;  **color**: **#fff**; } |

Codeabschnitt 192: Formatierung des Buttons auf der Startseite

Mit den Schlagwörtern *hover, before, active* etc. kann den Elementen oder auch Kategorien Effekte zugewiesen werden. Diese werden ausgeführt, wenn sich beispielsweise der Mauszeiger über dem entsprechenden Element befindet. Hierbei können dann Einstellungen im Bereich der Sichtbarkeit, Skalierung o.ä. angewendet werden.

|  |
| --- |
| .**nav-menu a**:**hover**:**before**, .**nav-menu li**:**hover** > **a**:**before**, .**nav-menu** .**menu-active** > **a**:**before** {  **visibility**: **visible**;  **-webkit-transform**: **scaleX**(1);  **transform**: **scaleX**(1); } |

Codeabschnitt 193: Effekte, wenn sich die Maus über einem Element befindet

Da heutzutage viele Menschen vorwiegend mit dem Smartphone anstatt mit dem PC oder Laptop im Internet surfen, kann mithilfe der Zusatzkategorie eine entsprechende angepasste Anzeige auf dem Bildschirm eines solchen Gerätes möglich gemacht werden. Die Einstellungsattribute sind dieselben.

|  |
| --- |
| **#mobile-nav** {  **position**: **fixed**;  **top**: 0;  **padding-top**: 18**px**;  **bottom**: 0;  **z-index**: 998;  **background**: **rgba**(52, 59, 64, 0.9);  **left**: -260**px**;  **width**: 260**px**;  **overflow-y**: **auto**;  **transition**: 0.4**s**; } |

Codeabschnitt 194: Navigationsleiste für Mobile-Ansicht

Nun werden noch die Effekte, welche sichtbar werden, wenn die einzelnen Kategorien in den Anzeigebereichen des Browsers gescrollt werden, mit dem Schlagwort *@Keyframes* und den Namen der Animation definiert. Danach werden die Elemente, welche von der rechten Seite erscheinen sollen, festgelegt.

|  |
| --- |
| **@keyframes fadeInUp** {  **from** {  **opacity**: 0;  **transform**: **translate3d**(0, 100%, 0);  }   **to** {  **opacity**: 1;  **transform**: **none**;  } }  .**fadeInUp** {  **animation-name**: **fadeInUp**; } |

Codeabschnitt 195: Programmierung der Effekte für die Startseite

* 1. **Log In / Sign Up**

Damit der Spiegel mithilfe der Website konfiguriert und individuell von dem Benutzer personalisiert werden kann, muss ein Benutzerkonto erstellt werden. Als Hilfestellung wurde ein Template von *Clever Techie* [[26]](#footnote-27)verwendet.

* + 1. Log In / Sign Up in HTML

In diesem Bereich des HTML-Dokuments werden drei Unterprogramme, welche in der Programmiersprache Javascript programmiert wurden, importiert.

|  |
| --- |
| <**head**>  <**title**>Sign-Up/Login Form</**title**>  **<?php include 'css/css.html'**; **?>** <**script src="js/validation.js" type="text/javascript"**></**script**>  <**script src="js/showpass.js" type="text/javascript"**></**script**>  <**script src="js/passmeter.js" type="text/javascript"**></**script**> </**head**> |

Codeabschnitt 196: Inkludieren der Dateien zum Überprüfen der Eingabe

Bevor sich ein Benutzer einloggen kann, muss er zuerst registriert werden. Hierfür müssen alle Felder, welche mit dem \* versehen sind, ausgefüllt werden. Für die Eingabe von den jeweiligen Informationen wird das Texteingabefeld von HTML verwendet, welches zusätzliche Funktionen wie Autovervollständigung bietet. Diese werden aber im Falle des Registrierungsprozesses deaktiviert. Damit das Passwort später bei der Eingabe nicht sichtbar ist, wird im Bereich des Eingabetyps anstatt des normalen Textes Passwort ausgewählt. Die Kategorie *signup\_message* dient der Anzeige von etwaigen Fehlern, die während des Registrierungsprozess auftreten können.

|  |
| --- |
| <**form id="registerform" name="registerform" onsubmit="return** *signup\_validation*();**" method="post" autocomplete="off"**>  <**div class="top-row"**>  <**div class="field-wrap"**>  <**label**>  First Name<**span class="req"**>\*</**span**>  </**label**>  <**input type="text" autocomplete="off" name='firstname' id='firstname'**/>  </**div**>  <**div class="field-wrap"**>  <**label**>  Last Name<**span class="req"**>\*</**span**>  </**label**>  <**input type="text" autocomplete="off" name='lastname' id='lastname'**/>  </**div**>  </**div**>  <**div class="field-wrap"**>  <**label**>  Email Address<**span class="req"**>\*</**span**>  </**label**>  <**input type="text" autocomplete="off" name='remail' id='remail'**/>  </**div**>  <**div class="field-wrap"**>  <**label**>  Password<**span class="req"**>\*</**span**>  </**label**>  <**input type="password" autocomplete="off" name='rpassword' id='rpassword'**/>  </**div**>  <**div class="field-wrap"**>  <**label**>  Repeat Password<**span class="req"**>\*</**span**>  </**label**>  <**input type="password" autocomplete="off" name='repeatpassword' id='repeatpassword'**/>  </**div**>  <**span id="signup\_message"**></**span**>  <**button type="submit" class="button button-block" name="register" id="register"**/>Sign Up</**button**> </**form**> |

Codeabschnitt 197: Registrierungsformular in HTML

Ist der Registrierungsprozess abgeschlossen, kann der neu angelegte Benutzer eingeloggt werden. Über die *onsubmit*-Funktion in HTML kann eine Javascript-Funktion zum Datenabgleich mit der Datenbank aufgerufen werden.

|  |
| --- |
| <**form id="loginform" name="loginform" onsubmit="return** *login\_validation*();**" method="post" autocomplete="off"**>  <**div class="field-wrap"**>  <**label**>  Email Address<**span class="req"**>\*</**span**>  </**label**>  <**input type="text" autocomplete="off" name="email" id="email"**/>  </**div**>  <**div class="field-wrap"**>  <**label**>  Password<**span class="req"**>\*</**span**>  </**label**>  <**input type="password" autocomplete="off" name="password" id="password"**/>  </**div**>  <**p class="forgot"**><**a href="forgot.php"**>Forgot Password?</**p**></**a**>  <**div class="checkbox"**>  </**div**>  <**span id="login\_message"**></**span**>  <**button class="button button-block" name="login" id="login"**/>Log In</**button**> </**form**> |

Codeabschnitt 198: Log In-Formular in HTML

* + 1. Log In / Sign Up in CSS

Wie schon auf der Startseite wird nun auch den Formularen mittels CSS ein Design hinzugefügt. Hierbei werden die Listeneinträge, welche einen schnellen Wechsel zwischen Log In- und Registrierungsformular ermöglichen, formatiert. Dies wird mit der *active* Funktion realisiert. Dabei wird die Hintergrundfarbe des Listeneintrags geändert.

|  |
| --- |
| .**tab-group** {  **list-style**: **none**;  **padding**: 0;  **margin**: 0 0 40**px** 0;  **background-color**:**rgb**(44,137,205) }  .**tab-group**:**after** {  **content**: **""**;  **display**: **table**;  **clear**: **both**; }  .**tab-group li a** {  **display**: **block**;  **text-decoration**: **none**;  **padding**: 15**px**;  **background-color**: **rgba**(160, 179, 176, 0.25);  **color**: **#a0b3b0**;  **font-size**: 20**px**;  **float**: **left**;  **width**: 50%;  **text-align**: **center**;  **cursor**: **pointer**;  **-webkit-transition**: .5**s ease**;  **transition**: .5**s ease**; }  .**tab-group li a**:**hover** {  **background-color**: **rgb**(44,137,205);  **color**: **#ffffff**; }  .**tab-group** .**active a** {  **background-color**: **rgb**(44,137,205);  **color**: **#ffffff**; }  .**tab-content** > **div**:**last-child** {  **display**: **none**; } |

Codeabschnitt 199: Gruppe zum Wechseln zwischen Registrierungs- und Log In-Formular

* + 1. Log In / Sign Up in Javascript

Um mögliche Hackerangriffe zu verhindern, wird die Eingabe des Benutzers mittels Javascript überprüft.

|  |
| --- |
| **function** *login\_validation*() {  **var** email = ***document***.forms[**"loginform"**][**"email"**].**value**;  **var** password = ***document***.forms[**"loginform"**][**"password"**].**value**;  **var** el;  **var** re = ^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/   **if** (email == **""**)  {  el = ***document***.getElementById(**'login\_message'**);  el.innerHTML = **"<div class='info-alert'>Email field is required!</div>"**;  **return false**;  }  **else if** (!re.test(email))  {  el = ***document***.getElementById(**'login\_message'**);  el.innerHTML = **"<div class='info-alert'>Please enter valid email!</div>"**;  **return false**;  }  **else if** (password == **""**)  {  el = ***document***.getElementById(**'login\_message'**);  el.innerHTML = **"<div class='info-alert'>Password field is required!</div>"**;  **return false**;  }  **else** {  **return true**;  } } |

Codeabschnitt 200: Überprüfung der Benutzereingabe

* + 1. Log In / Sign Up in PHP

Bevor nun die Funktion der Log In-Form in der Programmiersprache PHP programmiert wird, muss noch eine Verbindung zur Datenbank hergestellt werden.

In diesem Bereich des Codes wird mithilfe von PHP eine Verbindung zur Datenbank hergestellt, welche aktuell noch auf dem lokalen Rechner gehostet wird. Wenn die Verbindung zur Datenbank fehlgeschlagen hat, wird eine Fehlermeldung ausgegeben und der Verbindungsaufbau wird beendet. Auf diese Klasse wird anschließend von den anderen zugegriffen um die Verbindung zur Datenbank herzustellen.

|  |
| --- |
| **<?php** $host = **'localhost'**;  $user = **'root'**;  $password = **''**;   $mysqli = **new** mysqli($host,$user,$password);  **if** ($mysqli->**connect\_errno**)  {  *printf*(**"Connection failed: %s\n"**, $mysqli->**connect\_error**);  **die**();  }   **if** (!$mysqli->query(**'CREATE DATABASE accounts'**))  {  *printf*(**"Errormessage: %s\n"**, $mysqli->**error**);  } |

Codeabschnitt 201: Verbindungsherstellung zur Datenbank mit PHP

Hierbei wird überprüft, ob die E-Mail-Adresse noch in den Cookies gespeichert ist, wodurch der Benutzer noch eingeloggt wäre. Weiters werden die Eingabefelder auf ihre ordnungsgemäße Eingabe der benötigten Informationen überprüft. Diese Informationen werden mit folgendem Code überprüft. Wenn diese Bedingungen erfüllt sind wird der Registrierungsprozess von dem jeweiligen Benutzer in der Datenbank gestartet.

|  |
| --- |
| **<?php  require 'db.php'**;  *session\_start*();   **if** (**isset**($\_SESSION[**'logged\_in'**]) == 1)  {  *header*(**"location: profile.php"**);  }   **if** (**isset**($\_COOKIE[**'email'**]))  {  $\_SESSION[**'email'**] = $\_COOKIE[**'email'**];  $\_SESSION[**'logged\_in'**] = 1;  *header*(**"location: profile.php"**);  }   **if** (**isset**($\_POST[**'login'**]) && $\_POST[**'email'**] != **""** && $\_POST[**'password'**] != **""**)  {  **require 'login.php'**;  }  **elseif** (**isset**($\_POST[**'register'**]) && $\_POST[**'firstname'**] != **""** && $\_POST[**'lastname'**] != **""** && $\_POST[**'remail'**] != **""** && $\_POST[**'rpassword'**] != **""** && $\_POST[**'repeatpassword'**] != **""**)  {  **require 'error.php'**;  }  **else** {  **require 'register.php'**;  }  **?>** |

Codeabschnitt 202: Überprüfung ob Benutzer noch eingeloggt ist mit PHP

Im oberen Bereich des Skripts werden die Benutzereingaben von den Textfeldern importiert und anschließend mit der Funktion *escape\_string* in einen für SQL Statements angepassten String umgewandelt. Im weiteren Verlauf der Anwendung werden die bereits registrierten E-Mail-Adressen mit der Eingegebenen abgeglichen, um zu verhindern, dass zwei Benutzerkonten mit derselben E-Mail existieren. Sind die Adressen nicht gleich, so werden die Benutzerdaten in die Datenbank eingetragen und dem Benutzer wird mithilfe der *mailto*-Funktion in PHP ein Bestätigungslink zugesendet. Die Webanwendung ist jedoch auch ohne der Bestätigung in vollem Maße benutzbar, wie der folgende Code zeigt:

|  |
| --- |
| **<?php** $\_SESSION[**'email'**] = $\_POST[**'remail'**];  $\_SESSION[**'first\_name'**] = $\_POST[**'firstname'**];  $\_SESSION[**'last\_name'**] = $\_POST[**'lastname'**];   $first\_name = $mysqli->escape\_string($\_POST[**'firstname'**]);  $last\_name = $mysqli->escape\_string($\_POST[**'lastname'**]);  $email = $mysqli->escape\_string($\_POST[**'remail'**]);  $password = $mysqli->escape\_string(*password\_hash*($\_POST[**'rpassword'**], ***PASSWORD\_BCRYPT***));  $hash = $mysqli->escape\_string(*md5*(*rand*(0,1000)));   $result = $mysqli->query(**"SELECT** *\** **FROM users WHERE email='**$email**'"**) **or**  **die**($mysqli->error());   **if** ($result->**num\_rows** > 0) {  $\_SESSION[**'message'**] = **'<div class="info-alert">User with this email already exists!</div>'**;  *header*(**"location: error.php"**);  }  **else** {  $sql = **"INSERT INTO users (first\_name, last\_name, email, password, hash)"** .**"VALUES ('**$first\_name**','**$last\_name**','**$email**','**$password**', '**$hash**')"**;   **if** ($mysqli->query($sql)) {  $result = $mysqli->query(**"SELECT** *\** **FROM users WHERE email='**$email**'"**);  $user = $result->fetch\_assoc();   $id = $user[**'user\_id'**];   $\_SESSION[**'active'**] = 0;  $\_SESSION[**'logged\_in'**] = **true**;   **if** ($\_SESSION[**'email'**] == **'admin@admin.com'**) {  *header*(**"location: admin.php"**);  }   **else** {  $\_SESSION[**'message'**] =  **"<div class='info-success'>Welcome to Pinnovations!</div>"**;  $to = $email;  $headers = **'MIME-Version: 1.0'** . **"\r\n"**;  $headers .= **'Content-type: text/html; charset=iso-8859-1'** . **"\r\n"**;  $subject = **'Account Verification'**;  $message = **'<html>“Kopf und Inhalt der Nachricht mit Styling“**  **</html>'**;  *mail*($to, $subject, $message, $headers);  *header*(**"location: success.php"**);  }  }  **else** {  $\_SESSION[**'message'**] = **'<div class="info-alert">Registration failed!</div>'**;  *header*(**"location: error.php"**);  }  } **?>** |

Codeabschnitt 203: Registrierungsprozess in PHP

* 1. **Interner Bereich**

Im internen Bereich der Website hat der Benutzer die Möglichkeit, die Anordnung der Anwendungen auf dem Spiegel individuell zu gestalten und E-Mail sowie Kalender mit seinen persönlichen Zugangsdaten zu verbinden.

* + 1. Interner Bereich in HTML

Im Kopf-Bereich der Website befinden sich die Imports für die Navigationsleiste, welche im obersten Bereich der Website angezeigt wird. Weiters befinden sich dort auch Verweise auf Skripts für die Effekte, welche auf der Website genutzt werden. In der Listenkategorie *sortable* befinden sich die vier Felder, welche die Anwendungen am Spiegel repräsentieren. Jedes Listenelement steht für eine Anwendung auf dem Spiegel. Mithilfe des Buttons wird die derzeitige Anordnung als String an die Datenbank gesendet, wie der folgende Code zeigt:

|  |
| --- |
| **li class="ui-state-default" id="element1"**>**<?php echo** $alignment[0]; **?>**</**li**> <**li class="ui-state-default" id="element2"**>**<?php echo** $alignment[1]; **?>**</**li**> <**li class="ui-state-default" id="element3"**>**<?php echo** $alignment[2]; **?>**</**li**> <**li class="ui-state-default" id="element4"**>**<?php echo** $alignment[3]; **?>**</**li**> <**button class="submit" onclick="***postOrderToDatabase***" style="color**: **#ffff**; **background-color**: **rgb**(44,137,205); **border-radius**: 15**px**; **margin-top**: 40**px**; **height**: 50**px**; **width**: 300**px**; **text-align**: **center**;**"**>Submit Order</**button**> |

Codeabschnitt 204: Vier Anwendungen am Spiegel in HTML

* + 1. Interner Bereich in CSS

Im Bereich des Stylings wird lediglich ein Rahmen um die Listenelemente angebracht, um diese leichter verschiebbar zu gestalten. Zusätzlich dazu wurde der Mauszeiger so angepasst, dass der klassische *Move*-Zeiger, welcher im Explorer von Windows beim Verschieben von Dateien mit *Drag & Drop* sichtbar wird. Dies erleichtert dem Benutzer zu wissen, wann ein Element verschiebbar ist.

|  |
| --- |
| #sortable li {  cursor: move;  vertical-align: middle;  background-color: rgb(44,137,205);  border-radius: 20px;  color: rgb(255, 255, 255);  margin: 3px 3px 3px 3px;  padding: 10px;  float: left;  width: 150px;  height: 150px;  font-family: "Open Sans", sans-serif;  size: 1.7em;  text-align: center; } |

Codeabschnitt 205: Vier Anwendungen des Spiegels in CSS

* + 1. Interner Bereich in Javascript

Mittels Javascript wird die *Drag & Drop*-Funktion den einzelnen Elementen hinzugefügt. Die Reihenfolge verschiebt sich anschließend mit der *sortable*-Methode, da sobald sich ein Element über dem anderen befindet, alle davor befindlichen um eine Position nach vorne bewegen. Das Element, über welches sich das zu verschiebende befindet, bewegt sich eine Position nach hinten. Das Ablegen des Elements an der gewünschten Position wird mit der *disableSelection*-Funktion realisiert, wie der nachfolgende Code zeigt:

|  |
| --- |
| $( **function**() {  $( **"#sortable"** ).sortable();  $( **"#sortable"** ).disableSelection(); } ); |

Codeabschnitt 206: Drag and Drop Funktion in PHP

Um es dem Benutzer zu ermöglichen, die E-Mail bzw. die Kalenderanwendung am Spiegel mit den persönlichen Benutzerkonten zu verbinden, wird nach einem Mausklick auf dem entsprechenden Feld ein Dialogfenster geöffnet, wo E-Mail-Adresse und Passwort eingegeben werden können. Diese werden anschließend an die Datenbank als String gesendet. Mit dem folgenden Code wird geprüft, an welcher Position sich die E-Mail bzw. Kalenderanwendung aktuell befindet, um die Eingabe der Zugangsdaten bei genau diesen Feldern zu ermöglichen:

|  |
| --- |
| **var *element1*** = $(**"#element1"**).text(); $(***document***).ready(**function**(){  $(**"#element1"**).click(**function**(){  **if**($(**"#element1"**).text() == **"email"**){  **do** {  **var** email = *prompt*(**"Email"**,**""**);  email.str\_replace(**"//"**,**""**);  email.str\_replace(**"/"**, **""**);  email.str\_replace(**"\'"**, **""**);  email.str\_replace(**"<"**,**""**);  } **while** (email == **null** || email == **""**);   **do** {  **var** password = *prompt*(**"Password"**, **""**);  password.str\_replace(**"//"**,**""**);  password.str\_replace(**"/"**, **""**);  password.str\_replace(**"\'"**, **""**);  password.str\_replace(**"<"**, **""**);  } **while** (password == **null** || password == **""**);  **var** password = *prompt*(**"Password"**,**""**);  } **else if** ($(**"#element1"**).text() == **"calendar"**) {  **do** {  **var** email = *prompt*(**"Email"**,**""**);  email.str\_replace(**"//"**, **""**);  email.str\_replace(**"/"**,**""**);  email.str\_replace(**"\'"**, **""**);  email.str\_replace(**"<"**, **""**);  } **while** (email == **""** || email == **null**);  **do**{  **var** password = *prompt*(**"Password"**,**""**);  password.str\_replace(**"//"**, **""**);  password.str\_replace(**"/"**,**""**);  password.str\_replace(**"\'"**,**""**);  password.str\_replace(**"<"**, **""**);  }**while**(password == **null** || password == **""**);  }  }); }); |

Codeabschnitt 207: Benutzereingabe für E-Mail- und Kalender-Widget

* + 1. Interner Bereich PHP

Mit PHP wird nun geprüft, ob der Benutzer überhaupt berechtigt ist, den internen Bereich der Website zu besichtigen. Zusätzlich dazu wird mithilfe von PHP die aktuelle Anordnung der Widgets von der Datenbank abgefragt und anschließend die Felder dementsprechend ausgerichtet. Sobald die Anordnung verfügbar ist, wird diese mit der Funktion *explode* in einem Array abgespeichert, wie der nachfolgende Code zeigt:

|  |
| --- |
| **require 'db.php'**;  $sql = **"SELECT data from users"**;  *// $update = "UPDATE users SET data = '$order' WHERE id=1";* $result = $mysqli->query($sql); $widgets = **array**(**"weather"**,**"clock"**,**"calendar"**,**"email"**);  $output = *mysqli\_fetch\_assoc*($result); $alignment = *explode*(**"//"**,$output[**"data"**]); |

Codeabschnitt 208: Herunterladen der Anordnung von der Datenbank

1. **Ergebnis**

Zusammenfassend können wir mit Stolz behaupten, alle Musskriterien erfüllt zu haben und allen Hindernissen und Herausforderungen zum Trotz alles mehr oder weniger so realisiert haben, wie wir das im Vorfeld erhofft hatten. Hinzu kamen einige Erweiterungen, welche wir aus zeitlichen Gründen beginnen und größtenteils sogar abschließen konnten. Das Konzept unseres Projekts hat neben den von uns hinzugefügten noch viele zusätzliche Erweiterungsmöglichkeiten. Unter anderem können weitere Widgets wie Facebook, WhatsApp, YouTube oder Nachrichten hinzugefügt werden, wobei dann auswählbar werden könnte, welche Widgets am Spiegel angezeigt werden und welche nicht. Auch möglich ist, den Spiegel mit einer Gesichtserkennung abzusichern, sodass nur mehr ein Benutzer auf ein Profil zugreifen kann.



Abbildung 72: Finaler Test

1. **Abbildungsverzeichnis**

[Abbildung 1: Sketch des Komplettsystems 15](#_Toc510595743)

[Abbildung 2: Blockschaltbild Komplettsystem 16](#_Toc510595744)

[Abbildung 3: Grundrahmen 17](#_Toc510595745)

[Abbildung 4: Konstruktionsprozess des Rahmens 17](#_Toc510595746)

[Abbildung 5: Ausschnitt von der Rückseite des Rahmens 18](#_Toc510595747)

[Abbildung 6: Konstruierter Rahmen 18](#_Toc510595748)

[Abbildung 7: Rahmen in Archicad 19](#_Toc510595749)

[Abbildung 8: Einstellungen für Rendering 19](#_Toc510595750)

[Abbildung 9: Rendering Prozess 20](#_Toc510595751)

[Abbildung 10: Rahmen nach Rendering des Rahmens 20](#_Toc510595752)

[Abbildung 11: Rahmen nach dem durchgeführten Rendering 20](#_Toc510595753)

[Abbildung 12:Displayrahmen aus PVC 21](#_Toc510595754)

[Abbildung 13: Herstellerangaben Spiegel Glas (Quelle: Glas Schnabl) 21](#_Toc510595755)

[Abbildung 14: Hintere Abdeckung des Rahmens 22](#_Toc510595756)

[Abbildung 15: Aufhängung des Spiegels 22](#_Toc510595757)

[Abbildung 16:Zugriff auf Server mit Putty 23](#_Toc510595758)

[Abbildung 17: Bildschirm nach erfolgreichem Log In-Vorgang 24](#_Toc510595759)

[Abbildung 18: Updatevorgang des Servers 24](#_Toc510595760)

[Abbildung 19: Installationsvorgang von Apache-Webserver Pakete 25](#_Toc510595761)

[Abbildung 20: Testen der Apache Funktionalität 26](#_Toc510595762)

[Abbildung 21: Installation der MySQL Pakete am Server 27](#_Toc510595763)

[Abbildung 22: Ausgabe (test.php) 29](#_Toc510595764)

[Abbildung 23: PHPMyAdmin Oberfläche nach erfolgreicher Installation 31](#_Toc510595765)

[Abbildung 24: Verwaltung der bereits bestehenden Datenbanken 32](#_Toc510595766)

[Abbildung 25: Erzeugen einer neuen Tabelle in der Datenbank 32](#_Toc510595767)

[Abbildung 26: ID-Feld in der Tabelle 32](#_Toc510595768)

[Abbildung 27: Zusatzinformationen über den Benutzer 33](#_Toc510595769)

[Abbildung 28: Schnittstellen zwischen den Komponenten 34](#_Toc510595770)

[Abbildung 29: Struktur 35](#_Toc510595771)

[Abbildung 30: HTML-Interface 49](#_Toc510595772)

[Abbildung 31: Installationsprozess des hostapd-Paket 52](#_Toc510595773)

[Abbildung 32: Konfigurationsdatei der Netzwerke (wpa\_supplicant.conf) 58](#_Toc510595774)

[Abbildung 33: Struktur 61](#_Toc510595775)

[Abbildung 34: Konfigurationsmenü 62](#_Toc510595776)

[Abbildung 35: Pi4J Bibliothek 64](#_Toc510595777)

[Abbildung 36: Java Platform Manager 65](#_Toc510595778)

[Abbildung 37: Hinzufügen einer Java Platform 65](#_Toc510595779)

[Abbildung 38: Hinzufügen einer Java Platform 65](#_Toc510595780)

[Abbildung 39: Java Platform Manager 66](#_Toc510595781)

[Abbildung 40: Project Properties 66](#_Toc510595782)

[Abbildung 41: Hauptverzeichnis 67](#_Toc510595783)

[Abbildung 42: Bibliotheken-Ordner 69](#_Toc510595784)

[Abbildung 43: Project Properties 70](#_Toc510595785)

[Abbildung 44: E-Mail-Widget 93](#_Toc510595786)

[Abbildung 45: Kalender-Widget 95](#_Toc510595787)

[Abbildung 46: Wetter-Widget 97](#_Toc510595788)

[Abbildung 47: Uhren-Widget 100](#_Toc510595789)

[Abbildung 48: Hauptansicht 102](#_Toc510595790)

[Abbildung 49: E-Mail-App 107](#_Toc510595791)

[Abbildung 50: Erkennung der Handgeste in OpenCV 125](#_Toc510595792)

[Abbildung 51: APDS-9960 127](#_Toc510595793)

[Abbildung 52: Verdrahtung des Arduino UNO mit dem APDS-9960 128](#_Toc510595794)

[Abbildung 53: Test am „Serial Monitor“ 132](#_Toc510595795)

[Abbildung 54: Gestensensor im Einsatz 133](#_Toc510595796)

[Abbildung 55: Storyboard 134](#_Toc510595797)

[Abbildung 56: Struktur 136](#_Toc510595798)

[Abbildung 57: Benutzerdefinierte Klasse 137](#_Toc510595799)

[Abbildung 58: Benutzerdefinierte Klasse 138](#_Toc510595800)

[Abbildung 59: Implementierung des benutzerdefinierten Textfeldes 140](#_Toc510595801)

[Abbildung 60: Erstellen eines Kontos 140](#_Toc510595802)

[Abbildung 61: Anmelden 145](#_Toc510595803)

[Abbildung 62: Widgets zum verschieben 149](#_Toc510595804)

[Abbildung 63: Konto hinzufügen 155](#_Toc510595805)

[Abbildung 64: Konfiguration des WLAN-Netzwerkes 158](#_Toc510595806)

[Abbildung 65: Wählen der SDK 163](#_Toc510595807)

[Abbildung 66: Main\_Activity 164](#_Toc510595808)

[Abbildung 67: Sign\_Up\_Activity 168](#_Toc510595809)

[Abbildung 68: Widgets\_Activity 169](#_Toc510595810)

[Abbildung 69: Settings\_Activity 170](#_Toc510595811)

[Abbildung 70: Start-Ansicht im Emulator 171](#_Toc510595812)

[Abbildung 71: Navigationsleiste der Hauptseite 173](#_Toc510595813)

[Abbildung 72: Finaler Test 185](#_Toc510595814)

1. **Codeabschnitte**

[Codeabschnitt 1: Serverzugriff per SSH mit Linux/Mac-OS 23](#_Toc509558647)

[Codeabschnitt 2: Updatebefehl des Servers 24](#_Toc509558648)

[Codeabschnitt 3: Installieren der Updates 24](#_Toc509558649)

[Codeabschnitt 4: Installation Apache-Pakete 25](#_Toc509558650)

[Codeabschnitt 5: Installation von MariaDB am Server 26](#_Toc509558651)

[Codeabschnitt 6: Installation alternativ zu MariaDB von MySQL 26](#_Toc509558652)

[Codeabschnitt 7: Installation der Sicherheitsoptionen für MySQL/MariaDB 27](#_Toc509558653)

[Codeabschnitt 8: Zugriff auf MySQL-Datenbank mit Terminal 27](#_Toc509558654)

[Codeabschnitt 9: Schließen der Datenbankverbindung über das Terminal 28](#_Toc509558655)

[Codeabschnitt 10: Installation von PHP7.0-Paktete 28](#_Toc509558656)

[Codeabschnitt 11: Neustarten des Webservers 28](#_Toc509558657)

[Codeabschnitt 12: Wechsel in das Root-Verzeichnis des Webservers 28](#_Toc509558658)

[Codeabschnitt 13: Erstellung einer Testdatei zum Testen des Servers 28](#_Toc509558659)

[Codeabschnitt 14: Code (test.php) 28](#_Toc509558660)

[Codeabschnitt 15: Installation der MySQL Erweiterungen für PHP 29](#_Toc509558661)

[Codeabschnitt 16: Neustarten des Apache-Webservers 29](#_Toc509558662)

[Codeabschnitt 17: Benutzer erstellen 30](#_Toc509558663)

[Codeabschnitt 18: Verteilen von Zugriffsrechten 30](#_Toc509558664)

[Codeabschnitt 19: PhpMyAdmin installieren 31](#_Toc509558665)

[Codeabschnitt 20: Definition der Widgets 33](#_Toc509558666)

[Codeabschnitt 21: Allgemeine Konfigurationsdatei (config.php) 36](#_Toc509558667)

[Codeabschnitt 22: Konfigurationsdatei für die Kontoerstellung (signup\_config.php) 36](#_Toc509558668)

[Codeabschnitt 23: Konfigurationsdatei für die Anmeldung (signin\_config.php) 36](#_Toc509558669)

[Codeabschnitt 24: Konfigurationsdatei für Datenaktualisierungen (setdata\_config.php) 37](#_Toc509558670)

[Codeabschnitt 25: Konfigurationsdatei für das Laden von Daten (getdata\_config.php) 37](#_Toc509558671)

[Codeabschnitt 26: Klasse für Darstellung der Tabelle (Column.php) 38](#_Toc509558672)

[Codeabschnitt 27: Konstruktor (Database.php) 40](#_Toc509558673)

[Codeabschnitt 28: Funktion zum Verbinden der Datenbank (Database.php) 40](#_Toc509558674)

[Codeabschnitt 29: Funktion zum Schließen der Verbindung (Database.php) 41](#_Toc509558675)

[Codeabschnitt 30: Funktion zum Erstellen einer Tabelle (Database.php) 41](#_Toc509558676)

[Codeabschnitt 31: SQL-Statement zum Tabelle erstellen (Database.php) 42](#_Toc509558677)

[Codeabschnitt 32: Funktion zum Selektieren eines Datensatzes (Database.php) 42](#_Toc509558678)

[Codeabschnitt 33: SQL-Statement zum Selektieren eines Benutzers (Database.php) 43](#_Toc509558679)

[Codeabschnitt 34: Datensatz eines Benutzers (Database.php) 43](#_Toc509558680)

[Codeabschnitt 35: Funktion zum Selektieren eines Datensatzes (Database.php) 43](#_Toc509558681)

[Codeabschnitt 36: Funktion zum Aktualisieren eines Datensatzes (Database.php) 44](#_Toc509558682)

[Codeabschnitt 37: SQL-Statement zum Aktualisieren eines Datensatzes (Database.php) 44](#_Toc509558683)

[Codeabschnitt 38: Funktion zum Einfügen eines Benutzers (Database.php) 45](#_Toc509558684)

[Codeabschnitt 39: SQL-Statement zum Einfügen eines Benutzers (Database.php) 45](#_Toc509558685)

[Codeabschnitt 40: Schnittstelle zum Erstellen eines Kontos (signup.php) 46](#_Toc509558686)

[Codeabschnitt 41: Schnittstelle zum Anmelden (signin.php) 47](#_Toc509558687)

[Codeabschnitt 42: Schnittstelle zum Aktualisieren eines Datensatzes (setdata.php) 47](#_Toc509558688)

[Codeabschnitt 43: Schnittstelle zum Auswählen eines Datensatzes (getdata.php) 48](#_Toc509558689)

[Codeabschnitt 44: HTML-Interface zum Anmelden (interface\_signin.php) 49](#_Toc509558690)

[Codeabschnitt 45: Update des Raspberry Pi 50](#_Toc509558691)

[Codeabschnitt 46: Installation der Aktualisierungen am Raspberry Pi 50](#_Toc509558692)

[Codeabschnitt 47: Installation von PHP am Raspberry Pi 50](#_Toc509558693)

[Codeabschnitt 48: Installation Apache-Server am Raspberry Pi 51](#_Toc509558694)

[Codeabschnitt 49: Erstellung einer Testdatei zum Testen des Servers 51](#_Toc509558695)

[Codeabschnitt 50: Code zum Testen des Servers 51](#_Toc509558696)

[Codeabschnitt 51: Installation von hostapd am Raspberry Pi 52](#_Toc509558697)

[Codeabschnitt 52: Installation der dnsmasq-Pakete am Raspberry Pi 52](#_Toc509558698)

[Codeabschnitt 53: Deaktivierung des hostapd-Service 52](#_Toc509558699)

[Codeabschnitt 54: Deaktivierung des dnsmasq-Service 52](#_Toc509558700)

[Codeabschnitt 55: Öffnen der Konfigurationsdatei von hostapd 52](#_Toc509558701)

[Codeabschnitt 56: Inhalt der Konfigurationsdatei 53](#_Toc509558702)

[Codeabschnitt 57: Übergabe des Pfades der Konfigurationsdatei 53](#_Toc509558703)

[Codeabschnitt 58: Öffnen der Konfigurationsdatei von dnsmasq 53](#_Toc509558704)

[Codeabschnitt 59: Konfiguration des dnsmasq 54](#_Toc509558705)

[Codeabschnitt 60: Anzeigen der Netzwerkschnittstellen 54](#_Toc509558706)

[Codeabschnitt 61: Erste fünf Zeilen der Interfaces-Datei 54](#_Toc509558707)

[Codeabschnitt 62: Erstellung einer Sicherungsdatei der Netzwerkschnittstellen 54](#_Toc509558708)

[Codeabschnitt 63: Öffnen der Konfigurationsdatei für IP-Forwarding 55](#_Toc509558709)

[Codeabschnitt 64: Aktivierung von IP-Forwarding 55](#_Toc509558710)

[Codeabschnitt 65: Erstellung eines Autohotspot-Skripts 55](#_Toc509558711)

[Codeabschnitt 66: Inhalt Autohotspot-Skript 55](#_Toc509558712)

[Codeabschnitt 67: Starten des Autohotspot-Skriptes 55](#_Toc509558713)

[Codeabschnitt 68: Autohotspot.service 58](#_Toc509558714)

[Codeabschnitt 69: Erteilung der Benutzerrechte www-data 58](#_Toc509558715)

[Codeabschnitt 70: Wifi-Config-Interface in HTML 59](#_Toc509558716)

[Codeabschnitt 71: Wifi-Config.php 60](#_Toc509558717)

[Codeabschnitt 72: Mit dem Raspberry Pi verbinden 62](#_Toc509558718)

[Codeabschnitt 73: Java-Version abfragen 63](#_Toc509558719)

[Codeabschnitt 74: Löschen von OpenJDK 63](#_Toc509558720)

[Codeabschnitt 75: Hinzufügen des Digital-Keys 63](#_Toc509558721)

[Codeabschnitt 76: Hinzufügen des Digital-Keys 63](#_Toc509558722)

[Codeabschnitt 77: Installation von Java 8 63](#_Toc509558723)

[Codeabschnitt 78: Java-Version abfragen 63](#_Toc509558724)

[Codeabschnitt 79: Installation von Pi4J 64](#_Toc509558725)

[Codeabschnitt 80: Kernel Downgrade 64](#_Toc509558726)

[Codeabschnitt 81: Konfigurieren des X11-Servers (build.xml) 68](#_Toc509558727)

[Codeabschnitt 82: Voreinstellung zum Ausführen von GUI-Apps 68](#_Toc509558728)

[Codeabschnitt 83: Rotieren des Bildschirms (/boot/config.txt) 68](#_Toc509558729)

[Codeabschnitt 84: Konfigurationsdatei der Datenbank (DatabaseConfig.java) 70](#_Toc509558730)

[Codeabschnitt 85: Konfigurationsdatei der Dateienpfade (FileConfig.java) 71](#_Toc509558731)

[Codeabschnitt 86: Konfigurationsdatei der Schriftart (FontConfig.java) 71](#_Toc509558732)

[Codeabschnitt 87: Konfigurationsdatei der Bilder (ImageConfig.java) 71](#_Toc509558733)

[Codeabschnitt 88: Klasse des Tabellenschemas (Column.java) 72](#_Toc509558734)

[Codeabschnitt 89: Konstruktor der Datenbankschnittstelle (Database.java) 74](#_Toc509558735)

[Codeabschnitt 90: Funktion zum Verbindungsaufbau (Database.java) 74](#_Toc509558736)

[Codeabschnitt 91: Funktion zum Beenden der Verbindung (Database.java) 75](#_Toc509558737)

[Codeabschnitt 92: Funktion zum Selektieren eines Datensatzes (Database.java) 75](#_Toc509558738)

[Codeabschnitt 93: Funktion zum Laden eines Feldes (InterfaceDatabase.java) 76](#_Toc509558739)

[Codeabschnitt 94: Klasse des Terminschemas (Appointment.java) 77](#_Toc509558740)

[Codeabschnitt 95: Klasse des Uhrenschemas (Clock.java) 78](#_Toc509558741)

[Codeabschnitt 96: Klasse des Tageschemas (Day.java) 78](#_Toc509558742)

[Codeabschnitt 97: Klasse des Mailschemas (Email.java) 79](#_Toc509558743)

[Codeabschnitt 98: Klasse des Ortschemas (GeoLocation.java) 80](#_Toc509558744)

[Codeabschnitt 99: Widget-Anordnung 81](#_Toc509558745)

[Codeabschnitt 100: Funktion zum Laden der Widget-Anordnung (InterfaceDownload.java) 82](#_Toc509558746)

[Codeabschnitt 101: Funktion zum Laden der E-Mails (InterfaceDownload.java) 83](#_Toc509558747)

[Codeabschnitt 102: Funktion zum Laden der Kalendereinträge (InterfaceDownload.java) 84](#_Toc509558748)

[Codeabschnitt 103: Python-Skript zum Laden der Kalendereinträge (IcloudCalendar.py) 85](#_Toc509558749)

[Codeabschnitt 104: Installieren von PyIcloud 85](#_Toc509558750)

[Codeabschnitt 105: Funktion zum Formatieren der Termine (InterfaceDownload.java) 86](#_Toc509558751)

[Codeabschnitt 106: Funktion zum Laden des Ortes (InterfaceDownload.java) 86](#_Toc509558752)

[Codeabschnitt 107: Funktion zum Laden des Wetters (InterfaceDownload.java) 87](#_Toc509558753)

[Codeabschnitt 108: Funktion zum Laden der Uhrzeit (InterfaceDownload.java) 88](#_Toc509558754)

[Codeabschnitt 109: Funktion zum Auslesen eines Files (Filereader.java) 89](#_Toc509558755)

[Codeabschnitt 110: Funktion zum Auslesen der Schriftart (FontSupport.java) 89](#_Toc509558756)

[Codeabschnitt 111: Funktion zum Formatieren des Textes (TableSupport.java) 91](#_Toc509558757)

[Codeabschnitt 112: Funktion die einen TableCellRenderer zurückgibt (TableSupport.java) 91](#_Toc509558758)

[Codeabschnitt 113: Exception wenn ein Feld nicht gefunden wird (FieldNotFoundException.java) 91](#_Toc509558759)

[Codeabschnitt 114: Überklasse der Widgets (Widget.java) 93](#_Toc509558760)

[Codeabschnitt 115: Klasse des E-Mail-Widgets (EmailWidget.java) 94](#_Toc509558761)

[Codeabschnitt 116: Klasse des Kalender-Widgets (CalendarWidget.java) 97](#_Toc509558762)

[Codeabschnitt 117: Klasse des Wetter-Widgets (WeatherWidget.java) 99](#_Toc509558763)

[Codeabschnitt 118: Klasse des Uhren-Widgets (ClockWidget.java) 101](#_Toc509558764)

[Codeabschnitt 119: Konstruktor der Hauptansicht (MainView.java) 103](#_Toc509558765)

[Codeabschnitt 120: Funktion zum Zeichnen der Widgets (MainView.java) 104](#_Toc509558766)

[Codeabschnitt 121: Funktion zum Aktualisieren der Ansicht (MainView.java) 106](#_Toc509558767)

[Codeabschnitt 122: Klasse der E-Mail-Ansicht (EmailView.java) 108](#_Toc509558768)

[Codeabschnitt 123: Klasse der Kalender-Ansicht (CalendarView.java) 110](#_Toc509558769)

[Codeabschnitt 124: Klasse der Wetter-Ansicht (WeatherView.java) 111](#_Toc509558770)

[Codeabschnitt 125: Klasse der Uhren-Ansicht (ClockView.java) 112](#_Toc509558771)

[Codeabschnitt 126: Klasse des Hauptframes (MainFrame.java) 113](#_Toc509558772)

[Codeabschnitt 127: Hauptfunktion (ReflectonUI.java) 114](#_Toc509558773)

[Codeabschnitt 128: Voreinstellungen (gesture\_v4.py) 122](#_Toc509558774)

[Codeabschnitt 129: Erstellen des Bereichs, in welchem Geste erkennt werden soll (gesture\_v4.py) 122](#_Toc509558775)

[Codeabschnitt 130: Aufteilen des Farbbilds in Graustufen (gesture\_v4.py) 123](#_Toc509558776)

[Codeabschnitt 131: Erstellen einer klaren Unterscheidung zwischen hellen und dunklen Stellen (gesture\_v4.py) 123](#_Toc509558777)

[Codeabschnitt 132: Anzeigen des schwarz-weißen Resultats (gesture\_v4.py) 123](#_Toc509558778)

[Codeabschnitt 133: Erstellen einer Umrandung um größte weiße Stelle (gesture\_v4.py) 124](#_Toc509558779)

[Codeabschnitt 134: Zeichnen der Konturen der sich ergebenden Form (gesture\_v4.py) 124](#_Toc509558780)

[Codeabschnitt 135: Winkelberechnungen zu Erkennung von ausgestreckten Fingern (gesture\_v4.py) 124](#_Toc509558781)

[Codeabschnitt 136: Ausführen der der Handgesten entsprechenden Ausgabe (gesture\_v4.py) 125](#_Toc509558782)

[Codeabschnitt 137: Inkludierungen und Vordefinierungen (Gesture\_detection\_v8.ino) 129](#_Toc509558783)

[Codeabschnitt 138: Globale Variablen (Gesture\_detection\_v8.ino) 129](#_Toc509558784)

[Codeabschnitt 139: interruptRoutine (Gesture\_detection\_v8.ino) 129](#_Toc509558785)

[Codeabschnitt 140: Initialisierungsprozesse (Gesture\_detection\_v8.ino) 130](#_Toc509558786)

[Codeabschnitt 141: Ausführen der Ausgabe-Funktion bei aktivierter Interrupt-flag (Gesture\_detection\_v8.ino) 130](#_Toc509558787)

[Codeabschnitt 142: Ausgabe der Befehle entsprechend der benötigten Bedingungen (Gesture\_detection\_v8.ino) 131](#_Toc509558788)

[Codeabschnitt 143: Vordefinitionen und Konfigurationen (Config.swift) 136](#_Toc509558789)

[Codeabschnitt 144: CollectionView mit Text (TextCollectionViewCell.swift) 137](#_Toc509558790)

[Codeabschnitt 145: Übergang von rechts nach links (SegueToLeft.swift) 138](#_Toc509558791)

[Codeabschnitt 146: Benutzerdefiniertes Textfeld (DesignableTextField.swift) 139](#_Toc509558792)

[Codeabschnitt 147: ViewController in Memory geladen (SignUpViewController.swift) 141](#_Toc509558793)

[Codeabschnitt 148: Memory-Problemwarnung (SignUpViewController.swift) 142](#_Toc509558794)

[Codeabschnitt 149: Aktion bei SignUp-Button (SignUpViewController.swift) 143](#_Toc509558795)

[Codeabschnitt 150: Anordnung der Widgets 143](#_Toc509558796)

[Codeabschnitt 151: Aktion bei Back-Button (SignUpViewController.swift) 143](#_Toc509558797)

[Codeabschnitt 152: Wechseln auf andere Ansicht (SignUpViewController.swift) 144](#_Toc509558798)

[Codeabschnitt 153: Anzeigen der virtuellen Tastatur (SignUpViewController.swift) 145](#_Toc509558799)

[Codeabschnitt 154: ViewController in Memory geladen (SignInViewController.swift) 146](#_Toc509558800)

[Codeabschnitt 155: Memory-Problemwarnung (SignInViewController.swift) 146](#_Toc509558801)

[Codeabschnitt 156: Aktion bei SignIn-Button (SignInViewController.swift) 147](#_Toc509558802)

[Codeabschnitt 157: Aktion bei Join-Button (SignInViewController.swift) 147](#_Toc509558803)

[Codeabschnitt 158: Wechseln zu Hauptansicht (SignInViewController.swift) 148](#_Toc509558804)

[Codeabschnitt 159: Anzeigen der virtuellen Tastatur (SignInViewController.swift) 149](#_Toc509558805)

[Codeabschnitt 160: ViewController in Memory geladen (HomeViewController.swift) 150](#_Toc509558806)

[Codeabschnitt 161: Memory-Problemwarnung (HomeViewController.swift) 151](#_Toc509558807)

[Codeabschnitt 162: Aktion Settings-Button (HomeViewController.swift) 151](#_Toc509558808)

[Codeabschnitt 163: Vorbereitung des Überganges (HomeViewController.swift) 151](#_Toc509558809)

[Codeabschnitt 164: Drag&Drop-Geste (HomeViewController.swift) 152](#_Toc509558810)

[Codeabschnitt 165: Funktionen der CollectionView (HomeViewController.swift) 153](#_Toc509558811)

[Codeabschnitt 166: Pop-Up-Fenster für E-Mail- bzw. Kalender-Account (HomeViewController.swift) 155](#_Toc509558812)

[Codeabschnitt 167: Auf Account zugeschnittene Widgets 155](#_Toc509558813)

[Codeabschnitt 168: Laden der Benutzerdaten (HomeViewController.swift) 156](#_Toc509558814)

[Codeabschnitt 169: Lokales Speichern der Widget-Anordnung (HomeViewController.swift) 157](#_Toc509558815)

[Codeabschnitt 170: Aktualisierung der Daten (HomeViewController.swift) 158](#_Toc509558816)

[Codeabschnitt 171: ViewController in Memory geladen (WiFiViewController.swift) 159](#_Toc509558817)

[Codeabschnitt 172: Memory-Problemwarnung (WiFiViewController.swift) 160](#_Toc509558818)

[Codeabschnitt 173: Aktion Connect-Button (WiFiViewController.swift) 160](#_Toc509558819)

[Codeabschnitt 174: Aktion Back-Button (WiFiViewController.swift) 161](#_Toc509558820)

[Codeabschnitt 175: Vorbereitung für Übergang (WiFiViewController.swift) 161](#_Toc509558821)

[Codeabschnitt 176: Anzeigen der virtuellen Tastatur (WiFiViewController.swift) 161](#_Toc509558822)

[Codeabschnitt 177: Deklaration des „Wanna Join?“-Buttons (MainActivity.java) 165](#_Toc509558823)

[Codeabschnitt 178: Funktion für Aktion bei Click auf Button (MainActivity.java) 165](#_Toc509558824)

[Codeabschnitt 179: Senden der eingegebenen Accountdaten an Liste, Überprüfen auf Übereinstimmung (MainActivity.java) 166](#_Toc509558825)

[Codeabschnitt 180: Erstellen des DBHandlers (DBHandler.java) 166](#_Toc509558826)

[Codeabschnitt 181: Standardfunktionen der Datenbank (DBHandler.java) 167](#_Toc509558827)

[Codeabschnitt 182: Umwandlung der Datenbank in String-Array (DBHandler.java) 167](#_Toc509558828)

[Codeabschnitt 183: Prüfen der Eingaben auf Richtigkeit (SignUpActivity.java) 168](#_Toc509558829)

[Codeabschnitt 184: Ausgeben der Datenbank-Einträge (SignUpActivity.java) 169](#_Toc509558830)

[Codeabschnitt 185: Funktion zum Verschieben der Widgets (WidgetsActivity.java) 170](#_Toc509558831)

[Codeabschnitt 186: Header-Section in HTML 173](#_Toc509558832)

[Codeabschnitt 187: Pinnovations Schriftzug in der Navigationsleiste 173](#_Toc509558833)

[Codeabschnitt 188: Übergangseffekte auf der Startseite 173](#_Toc509558834)

[Codeabschnitt 189: Formatierung Body-Tag in CSS 174](#_Toc509558835)

[Codeabschnitt 190: Formatierung selbstdefinierter Kategorie in CSS 174](#_Toc509558836)

[Codeabschnitt 191: Angleichung der Formatierung für Logo und Navigationsleiste 174](#_Toc509558837)

[Codeabschnitt 192: Formatierung des Buttons auf der Startseite 175](#_Toc509558838)

[Codeabschnitt 193: Effekte, wenn sich die Maus über einem Element befindet 175](#_Toc509558839)

[Codeabschnitt 194: Navigationsleiste für Mobile-Ansicht 175](#_Toc509558840)

[Codeabschnitt 195: Programmierung der Effekte für die Startseite 176](#_Toc509558841)

[Codeabschnitt 196: Inkludieren der Dateien zum Überprüfen der Eingabe 176](#_Toc509558842)

[Codeabschnitt 197: Registrierungsformular in HTML 177](#_Toc509558843)

[Codeabschnitt 198: Log In-Formular in HTML 178](#_Toc509558844)

[Codeabschnitt 199: Gruppe zum Wechseln zwischen Registrierungs- und Log In-Formular 178](#_Toc509558845)

[Codeabschnitt 200: Überprüfung der Benutzereingabe 179](#_Toc509558846)

[Codeabschnitt 201: Verbindungsherstellung zur Datenbank mit PHP 180](#_Toc509558847)

[Codeabschnitt 202: Überprüfung ob Benutzer noch eingeloggt ist mit PHP 180](#_Toc509558848)

[Codeabschnitt 203: Registrierungsprozess in PHP 181](#_Toc509558849)

[Codeabschnitt 204: Vier Anwendungen am Spiegel in HTML 182](#_Toc509558850)

[Codeabschnitt 205: Vier Anwendungen des Spiegels in CSS 182](#_Toc509558851)

[Codeabschnitt 206: Drag and Drop Funktion in PHP 183](#_Toc509558852)

[Codeabschnitt 207: Benutzereingabe für E-Mail- und Kalender-Widget 184](#_Toc509558853)

[Codeabschnitt 208: Herunterladen der Anordnung von der Datenbank 184](#_Toc509558854)

1. **Quellenverzeichnis**

Avago Technologies. (2013). *https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf.*

BootstrapMade. (2018). *https://bootstrapmade.com/demo/Regna/*.

Clever Techie. (2017). *https://clevertechie.com/php/20/login-system-php-mysql-database*.

Digi-Key-Electronics. (2016). *https://www.digikey.at/de/product-highlight/m/microchip-technology/mgc3030-mgc3130-3d-gesture-controllers*.

Ervideira, D. (2018). *https://github.com/dvdme/forecastio-lib-java/tree/master/src/com/github/dvdme/ForecastIOLib*.

Hedley, J. (2018). *https://jsoup.org/download*.

Le, J. (2016). *http://www.mediafire.com/file/rlylyu7265q6o63/handGesture.zip*.

Leap Motion Incorporated. (2017). *https://www.leapmotion.com/product/vr/#113*.

MaxMind Inc. (2018). *http://www.java2s.com/Code/Jar/g/Downloadgeoipapi1210jar.htm*.

MaxMind Inc. (2018). *https://dev.maxmind.com/geoip/legacy/geolite/*.

Merry, C. (2016). *https://www.raspberrypi.org/products/pi-noir-camera-v2/*.

OpenCV. (2017). *https://opencv.org/about.html*.

Oracle Corporation. (2018). *http://www.oracle.com/technetwork/java/javamail/javamail145-1904579.html*.

Oracle Corporation. (2018). *https://dev.mysql.com/downloads/connector/j/*.

Pi4J. (2018). *http://pi4j.com/download.html*.

Pimoroni GmbH. (2017). *https://shop.pimoroni.de/products/skywriter-hat*.

roboberry. (2017). *http://www.raspberryconnect.com/network/item/330-raspberry-pi-auto-wifi-hotspot-switch-internet.*

Sharp Corporated. (2014). *www.sharp.co.jp/products/device/doc/opto/gp2y0a21yk\_e.pdf.*

SparkFun Electronics. (2017). *https://www.sparkfun.com/products/12787*.

SparkFun Electronics. (2017). *https://www.sparkfun.com/products/13162*.

Sternberg, R. (2018). *https://github.com/ralfstx/minimal-json/releases*.

Sweigert, A. (2011). *http://coffeeghost.net/2011/05/09/moosegesture-python-mouse-gestures-module/*.

Tau, N. (2018). *https://github.com/demonnico/CHTCollectionViewWaterfallLayout*.

Tau, N. (2018). *https://github.com/demonnico/CHTCollectionViewWaterfallLayout*.

W3Schools. (2018). *https://www.w3schools.com/lib/w3.css.*

1. **Begleitprotokoll**

Themenstellung: Reflecton – Entwicklung eines intelligenten Spiegels

Kandidatinnen / Kandidaten: Hennermann Marco, Jeitler-Stehr Alexander,  
 Korntner Lukas

Jahrgang: 5CHEL

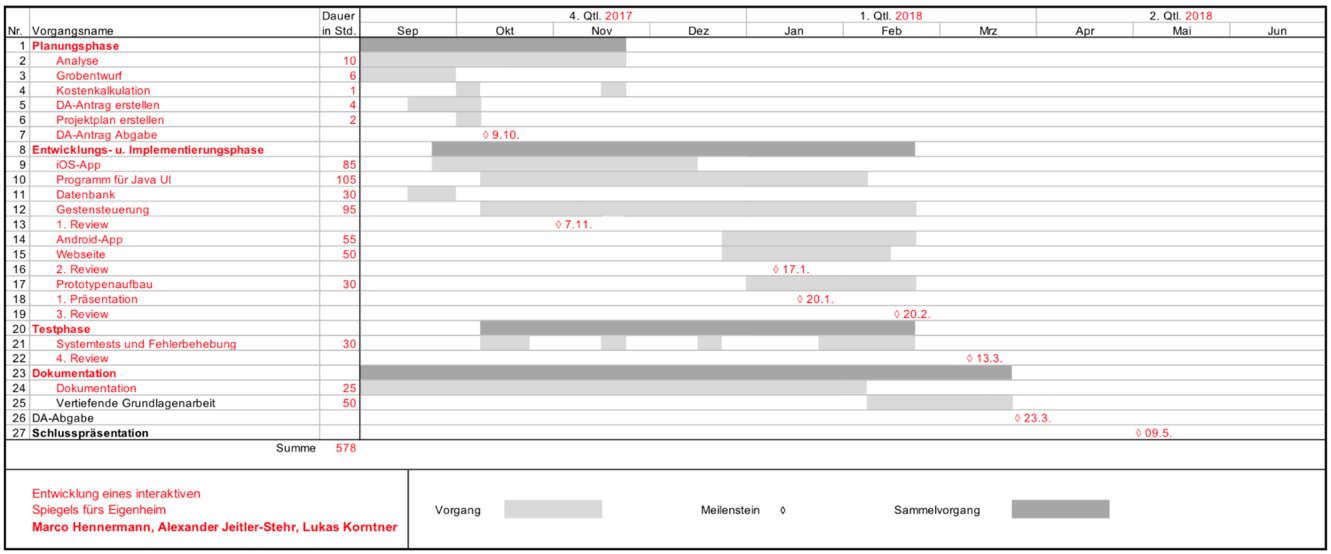
Betreuerin / Betreuer: AV Ing. Dipl.-Ing. Karl Heinz Steiner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Datum** | **Dauer** | **Personen** | **Thema** | **Anmerkung, Tasks** |
| 21.09.2017 | 180 min | HenM, JeiA, KorL, SteK | Teamfindung und Arbeitsaufteilung |  |
| 28.09.2017 | 120 min | HenM, JeiA, KorL, SteK | DA-Antrag |  |
| 03.10.2017 | 120 min | HenM, JeiA, KorL, SteK | 1.Review |  |
| 30.11.2017 | 60 min | HenM, JeiA, KorL, SteK | Gestaltung der Präsentation |  |
| 05.12.2017 | 120 min | HenM, JeiA, KorL, SteK | 2. Review |  |
| 13.01.2018 | 120 min | HenM, JeiA, KorL, SteK | Vorbereitung für Tag der offenen Tür |  |
| 18.01.2018 | 120 min | HenM, JeiA, KorL, SteK | 3. Review |  |
| 03.02.2018 | 180 min | HenM, JeiA, KorL, SteK | Analyse des Gesamtprodukts |  |
| 10.02.2018 | 120 min | HenM, JeiA, KorL, SteK | Erarbeiten der Inhalte der Diplomarbeit |  |
| 13.03.2018 | 120 min | HenM, JeiA, KorL, SteK | 4. Review |  |
| 15.03.2018 | 120 min | HenM, JeiA, KorL, SteK | Rückmeldung zu Diplomarbeitsfortschritt |  |
| 23.03.2018 | 60 min | HenM, JeiA, KorL, SteK | Fertigstellung der Diplomarbeit |  |
| Summe | 24 h |  |  |  |

1. **Anhang**
   1. **Kostenschätzung**

|  |  |  |
| --- | --- | --- |
| **Bezeichnung** | **Preis** | **Anzahl** |
| Raspberry Pi 3 | € 35,00 | 1 |
| SD-Karte 8 GB | € 8,00 | 1 |
| Halbdurchlässiges Spiegelglas | € 0,00 | 1 |
| 17“-LCD-Display plus Display-Controller | € 120,00 | 1 |
| 12V-Netzteil | € 11,00 | 1 |
| Infrarotsensor | € 5,00 | 1 |
| Schaumstoff | € 12,00 | 1 |
| Arduino UNO | € 21,00 | 1 |
| Gesamt | € 212,00 |

* 1. **Gantt-Diagramm**



1. W3Schools (2018): Library.https://www.w3schools.com/lib/w3.css, Abfrage am 21.03.2018 [↑](#footnote-ref-2)
2. Roboberry (2017): Raspberry Pi - Auto WiFi Hotspot Switch Internet. http://www.raspberryconnect.com/network/item/330-raspberry-pi-auto-wifi-hotspot-switch-internet, Abfrage am 22.03.2018 [↑](#footnote-ref-3)
3. MaxMind Inc. (2018): GeoLite Legacy Downloadable Database. https://dev.maxmind.com/geoip/legacy/geolite/, Abfrage am 21.03.2018 [↑](#footnote-ref-4)
4. Oracle Corporation (2018): Download Connector/J. https://dev.mysql.com/downloads/connector/j/, Abfrage am 21.03.2018 [↑](#footnote-ref-5)
5. Pi4J (2018): Download. http://pi4j.com/download.html, Abfrage am 21.03.2018 [↑](#footnote-ref-6)
6. Ervideira, David (2018): ForecastIOLib. https://github.com/dvdme/forecastio-lib-java/tree/master/src/com/github/dvdme/ForecastIOLib, Abfrage am 21.03.2018 [↑](#footnote-ref-7)
7. MaxMind Inc. (2018): Download. http://www.java2s.com/Code/Jar/g/Downloadgeoipapi1210jar.htm, Abfrage am 21.03.2018 [↑](#footnote-ref-8)
8. Oracle Corporation (2018): Downloads. http://www.oracle.com/technetwork/java/javamail/javamail145-1904579.html, Abfrage am 21.03.2018 [↑](#footnote-ref-9)
9. Hedley, Jonathan (2018): Download and install jsoup. https://jsoup.org/download, Abfrage am 21.03.2018 [↑](#footnote-ref-10)
10. Sternberg, Ralf (2018): minimal-json. https://github.com/ralfstx/minimal-json/releases, Abfrage am 21.03.2018 [↑](#footnote-ref-11)
11. Digi-Key Electronics (2016): Controller MGC3030/MGC3130 zur 3D-Gestensteuerung. <https://www.digikey.at/de/product-highlight/m/microchip-technology/mgc3030-mgc3130-3d-gesture-controllers>, Abfrage am 15. Oktober 2017 [↑](#footnote-ref-12)
12. Leap Motion Incorporated (2017): Let's get surreal. https://www.leapmotion.com/product/vr/#113, Abfrage am 15. Oktober 2017 [↑](#footnote-ref-13)
13. Pimoroni GmbH (2017): Skywriter HAT /PIM058. https://shop.pimoroni.de/products/skywriter-hat, Abfrage am 16. Oktober 2017 [↑](#footnote-ref-14)
14. SparkFun Electronics (2017): ZX Distance and Gesture Sensor. https://www.sparkfun.com/products/13162, Abfrage am 16. Oktober 2017 [↑](#footnote-ref-15)
15. SWEIGERT, Al (2011): MooseGesture 0.9.0. http://coffeeghost.net/2011/05/09/moosegesture-python-mouse-gestures-module/, Abfrage am 22. Oktober 2017 [↑](#footnote-ref-16)
16. ABELES, Peter (2017): Main Page. https://boofcv.org/index.php?title=Main\_Page, Abfrage am 21. Oktober 2017 [↑](#footnote-ref-17)
17. OpenCV team (2017): About. https://opencv.org/about.html, Abfrage am 22. Oktober 2017 [↑](#footnote-ref-18)
18. MERRY, Caitlyn (2016): PI NoIR camera V2. https://www.raspberrypi.org/products/pi-noir-camera-v2/, Abfrage am 22.10.2017 [↑](#footnote-ref-19)
19. LE, Jacky (2016): handGesture. http://www.mediafire.com/file/rlylyu7265q6o63/handGesture.zip, Abfrage am 23.10.2017 [↑](#footnote-ref-20)
20. SparkFun Electronics (2017): SparkFun RGB and Gesture Sensor - APDS-9960. https://www.sparkfun.com/products/12787, Abfrage am 14. November 2017 [↑](#footnote-ref-21)
21. Avago Technologies (2013): apds9960. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/apds9960.pdf, Abfrage am 28. November 2017 [↑](#footnote-ref-22)
22. Gp2y0a21yk0f www.sharp.co.jp/products/device/doc/opto/gp2y0a21yk\_e.pdf, Abfrage am 19. Dezember 2017 [↑](#footnote-ref-23)
23. Tau, Nicholas (2018): CHTCollectionViewWaterfallLayout. https://github.com/demonnico/CHTCollectionViewWaterfallLayout, Abfrage am 21.03.2018 [↑](#footnote-ref-24)
24. Tau, Nicholas (2018): CHTCollectionViewWaterfallLayout. https://github.com/demonnico/CHTCollectionViewWaterfallLayout, Abfrage am 21.03.2018 [↑](#footnote-ref-25)
25. BootstrapMade (2018): Welcome to Regna. https://bootstrapmade.com/demo/Regna/, Abfrage am 22.03.2018 [↑](#footnote-ref-26)
26. Clever Techie (2017): PHP Login System. https://clevertechie.com/php/20/login-system-php-mysql-database, Abfrage am 22.03.2018 [↑](#footnote-ref-27)