Projektbericht **Unison**

PROLOG

Heutzutage gibt es unzählige Technologien, die den Menschen dabei unterstützen, bestimmte Dinge im Leben leichter und schneller zu bewältigen. Einige dieser Technologien braucht man mehr als andere. Viele beschäftigen sich mit der Gesundheit des Menschen, überprüfen seinen Lebenshaushalt und sorgen für eine ausgewogene Ernährung oder einen optimalen Aktivitätsanteil. Mit Hilfe von Sensorik wird die Welt in Daten überführt. Sie übersetzen die gefühlte Qualität der physikalischen und chemischen Eigenschaften in quantitative Größen – sie bilden die Grundlage eines jeden Human Computer Interface.

Diese Thematik stand im dritten Semester an der Hochschule Osnabrück im Studiengang Media und Interaction Design im Fokus zweier Module. Unter der Diverse „Schneller, höher und weiter“ galt es ein intelligentes Wearable Device zu entwickeln, welches in Kommunikation mit einem Smartphone, einen selbstdefinierten Aspekt des täglichen Lebens effizienter gestaltet und bei der Bewältigung dessen unterstützt.

Zur Überführung des hier entstehenden Konzepts in einen funktionsfähigen Prototyp, hat eine Kooperation zwischen zwei Modulen stattgefunden.

Im Modul Interaction Design 2 stand die Realisierung des Wearable Devices im Fokus. Hier galt es mit Hilfe eines Arduinos die relevanten Daten sensorisch zu messen und eine entsprechende Bluetooth Kommunikation zu erschaffen, über welche die ermittelten Daten an das Smartphone übergeben werden können.

Mit Hilfe einer zugehörigen App, die es im Modul Webtechnologien 2 zu entwickeln galt, sollten die gesendeten Daten verarbeitet und in eine für den Nutzer verständliche und interpretierbare Form überführt werden.

Im Folgenden wird zunächst auf die allgemeine Konzeptionphase eingegangen. Im Anschluss wird auf die beiden oben aufgeführten Module differenziert in zwei separaten Kapiteln eingegangen. Hierbei wird zunächst die Konzeption und die Umsetzung des Wearable Devices aus dem Modul Interaction Design 2 thematisiert. Im Anschluss folgt eine genaue Beschreibung der Konzeption und der Realisierung der zugehörigen App aus dem Modul Webtechnologien 2.

KONZEPTION
DREI IDEEN

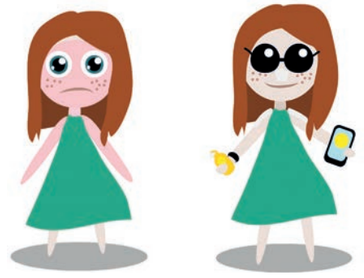
Der Hauptbestandteil der Konzeptionsphase bestand darin, einen Bereich aus dem täglichen Leben zu identifizieren, der in seiner derzeitigen Form als nicht optimal empfunden wird. Um sich diesem Bereich zu nähern, wurden zunächst Fragen und Probleme definiert, die einen im Alltag beschäftigen und dessen Lösung mit Hilfe eines Wearable Device herbeigeführt und vereinfacht werden kann. Hieraus resultierten drei wesentliche Probleme, zu denen in einem weiteren Schritt entsprechende Lösungsansätze konzipiert wurden.

Zwei der Probleme wurden im Bereich Gesundheit gesehen. Das erste beschäftigt sich speziell mit der Rückenhaltung. Vor allem Menschen, die viel Zeit im Büro vor einem Computer bzw. vor einem Laptop verbringen, nehmen oft unbemerkt eine ungesunde Haltung ein. Mit Hilfe eines Wearable Device soll diese durchgehend überprüft und gemessen werden. Sobald der Nutzer eine Zeit lang eine rückenbelastende Haltung einnimmt, warnt ihn das Wearable und fordert ihn durch ein Vibrationsfeedback auf, die Haltung zu wechseln. Weiterhin soll mit Hilfe einer zugehörigen App der Verlauf der Haltung aufgezeichnet und visualisiert werden, damit dem Nutzer ersichtlich ist, an welchen Tagen und bei welchen Tätigkeiten sein Rücken besonders belastet wird. Auch passende Rückentrainings sollen in der App zur Verfügung stehen, die mögliche Verspannungen des Rückens lösen.

Das zweite Problem behandelte das Thema UV-Strahlung der Sonne und dessen Verträglichkeit für die Haut. Viele Menschen bemerken nicht, wenn sie zu lange in der Sonne sind. Dies ist vor allem bei Menschen mit empfindlicher Haut problematisch, bei denen eine direkte und lange Sonneneinstrahlung zu starken Verbrennungen und Hautschäden führen kann. Um diesem Problem entgegen zu wirken, soll ein Armband zum Einsatz kommen, welches sensorisch die UV-Strahlung misst und diese Werte an eine zugehörige App übermittelt. Diese überprüft den zuvor eingetragenen Hauttyp mit der Dauer und der Intensität der Sonneneinstrahlung und informiert den Nutzer, dass ein erneutes Auftragen von Sonnencreme erforderlich ist. Des Weiteren zeigt die App durch die Kommunikation mit einer Wetter API besonders kritische Sonnentage für den Nutzer an.

Beide zuvor konzipierten Probleme wurden dem Bereich Gesundheit zugeschrieben, in welchem die Forschung schon weit vorangeschritten ist. So gibt es bereits diverse Ansätze, in denen ein Wearable Device in Kommunikation mit einer App versucht, Rückenschäden oder Hautprobleme durch Sonneneinstrahlung vorzubeugen.

Aus diesem Grunde fiel der Fokus auf den Bereich Life-Style und Mode. Das hier identifizierte Problem befasst sich mit der Findung und Zusammenstellung von Kleidungsstücke und Outfits, die farblich zu 100% übereinstimmen. Durch den Einsatz eines Wearable Devices, welches das Einscannen einer Farbe ermöglichen soll, sollten die Farbwerte mit Hilfe einer dazugehörigen App mit farbigen Kleidungsstücken verglichen und auf ihre Übereinstimmung überprüft werden können. Dieses Konzept wurde im Weiteren intensiver betrachtet und ausgearbeitet.



01 |

02 |

03 |

01 | **Sunnar**
02 | **Vitarity**
03 | **Colorpicker**

KONZEPTION
UNISON

Zielgruppe
Der oben aufgeführte Ansatz spricht wahrscheinlich weibliche Personen zwischen ca. 16 – 40 Jahren an, die sich mit Mode beschäftigen und viel Wert auf ein stimmiges, perfekt aufeinander abgestimmtes Outfit legen. Jedoch zählen auch speziell Mode- und Costumdesigner mit zur Zielgruppe, in deren Bereich das Thema Mode im Fokus ihrer Arbeit steht. Auch für Personen mit Farbfehlsichtigkeit kann das hier geplante Projekt mit dem Titel „Unison“ hilfreich sein, um zwischen verschiedenen Farbnuancen differenzieren zu können, wenn ihre Augen ihnen nicht die Möglichkeit hierzu bieten.

Problembeschreibung
Die Eine oder Andere kennt vielleicht das Problem: Man sucht farblich passende Schuhe und Schmuckstücke zu einem Kleid, welches aus einem dunklen Blaufarbton mit einen Stich ins Lila besteht. Sobald man Schuhe oder Schmuckstücke in einem Blauton gefunden hat, ist man sich nicht mehr sicher, ob dieser Farbton tatsächlich der des Kleides ist. Oder gibt es womöglich kleine Farbunterschiede, die dazu führen, dass das gesamte Outfit nicht mehr zusammen passt und sich die Farben beißen? Nicht nur die Erinnerung an die Farbe kann einen hier trügen, auch die Lichtverhältnisse im Laden und zu Hause spielen eine große Rolle bei der Farbwahrnehmung.

Lösungsansatz
Unison behandelt genau dieses Problem und erzielt eine einfache Lösung, damit man nicht etwa seinen gesamten Kleiderschrank zum Shoppen mitnehmen muss, um die Kleidungsstücke mit anderen Stücken vergleichen zu können. Unison ermöglicht es, Farben mit Hilfe eines Wearables neutral einzuscannen und diese in einer zugehörigen App, in Kombination mit einem Foto des Kleidungsstückes, zu speichern. So bietet Unison die Möglichkeit seinen eigenen Kleiderschrank in die App zu überführen und die Kleidungsstücke unter gleichen Lichtbedingungen im Geschäft mit anderen Kleidungsstücken hinsichtlich der Farben zu vergleichen.

Damit eine neutrale Farbmessung gewährleistet ist, besitzt das Wearable Device einen Farbsensor mit einer integrierten LED. Auf diese Art und Weise werden die Kleidungsstücke immer unter gleichen Lichtverhältnissen eingescannt. Dadurch soll ein sehr genauer Farbvergleich gewährleistet werden. Mit Hilfe eines Bluetooth-Moduls werden die gemessenen Werte an das Smartphone übermittelt, die dort gespeichert und mit anderen Werten verglichen werden können.

Nach und nach kann über die App ein eigener virtueller Kleiderschrank mit entsprechenden Fotos, Beschreibungstexten, Tags und mit den zugehörigen Farbwerten erstellt werden. Bei der Suche nach den passenden Kleidungsstücken bietet Unison die Möglichkeit, über die App, die im Geschäft eingescannten Farben, mit einem oder mehreren ausgewählten Kleidungsstücken aus dem virtuellen Kleiderschrank (Farbschrank), zu vergleichen. Eine detaillierte Übersicht informiert den Nutzer im Anschluss über die Übereinstimmung der RGB-Werte in Prozent.

Nicht nur die Stücke aus dem Kleiderschrank, sondern auch die eingescannte Farben beim Einkaufen können in einer Art „Shopping List“ gespeichert werden. So erhält der Benutzer die Chance die Items zu einem späteren Zeitpunkt mit anderen Kleidungsstücken zu vergleichen oder sich noch einmal über den Ort bzw. den Preis bestimmter Kleidungsstücke zu informieren.

INTERACTION DESIGN 2

Im Rahmen des Moduls „Interaction Design 2“ wurde im dritten Semester der Schwerpunkt der Veranstaltung auf die technische Bereiche gelenkt, die für die Erschaffung eines Wearable Devices unerlässlich sind. Hierbei wurden nicht nur elementare Themen, wie die Elektrotechnik behandelt, sondern auch ein Überblick über die vielfältige Sensorik, mit dessen Hilfe die Welt in messbare Werte überführt wird, gegeben.

Mit Hilfe dieser Grundlagen sollte nun, für das oben aufgeführte Konzept Unison, ein Wearable Device entwickelt werden, welches mit Hilfe von Sensoren die für das Projekt benötigen Werte ermittelt.

Im Folgenden soll zunächst auf die technische Konzeption des Wearables eingegangen werden, bevor im Anschluss eine detaillierte Erläuterung der Umsetzung stattfindet. Hierbei erfolgt eine genaue Beschreibung der Vorgehensweise, wobei explizit Probleme und Erkenntnisse erläutert werden. Abschließend wird die Planung und Herstellung des zugehörigen Gehäuses thematisiert und das letztendliche Ergebnis präsentiert.

04 | **Unison Wearable Device** im Einsatz

WEARABLE DEVICE
TECHNIK

Bevor mit der Konzeption des äußeren Erscheinungsbildes des Wearables begonnen werden konnte, wurde der Fokus auf die technische Realisierung gelegt. Der Anspruch bestand darin, ein Wearable zu schaffen, welches Farben unter neutralem Licht einscannen kann und die ermittelten Werte automatisch an ein Smartphone übermittelt.

Jedoch sollte nicht nur diese Hauptfunktion berücksichtigt werden, sondern auch andere Parameter, die für den Umgang mit den Wearable von Nutzen sind. So sollte der Benutzer die Möglichkeit besitzen das Gerät an- oder auszuschalten und es mit Hilfe eines USB-Gerätes aufladen zu können. Des Weiteren sollte der Benutzer über ein haptisches Feedback darüber informiert werden, sobald er eine Farbe erfolgreich eingescannt hat. Auch der Aspekt des Handlings spielte eine große Rolle bei der Planung. Ein Gerät welches die Aufgabe besitzt Farben einzuscannen und einem bei der Wahl seines Outfittes zu unterstützen, sollte der Benutzer jederzeit bei sich haben. Um dies zu gewährleisten, sollte das Gerät so klein und leicht wie möglich gehalten werden.

All diese Dinge mussten vor der eigentlichen Umsetzung bedacht werden, um später berücksichtigt werden zu können.

05 | **Technische Umsetzung**
Arduino Mini Pro

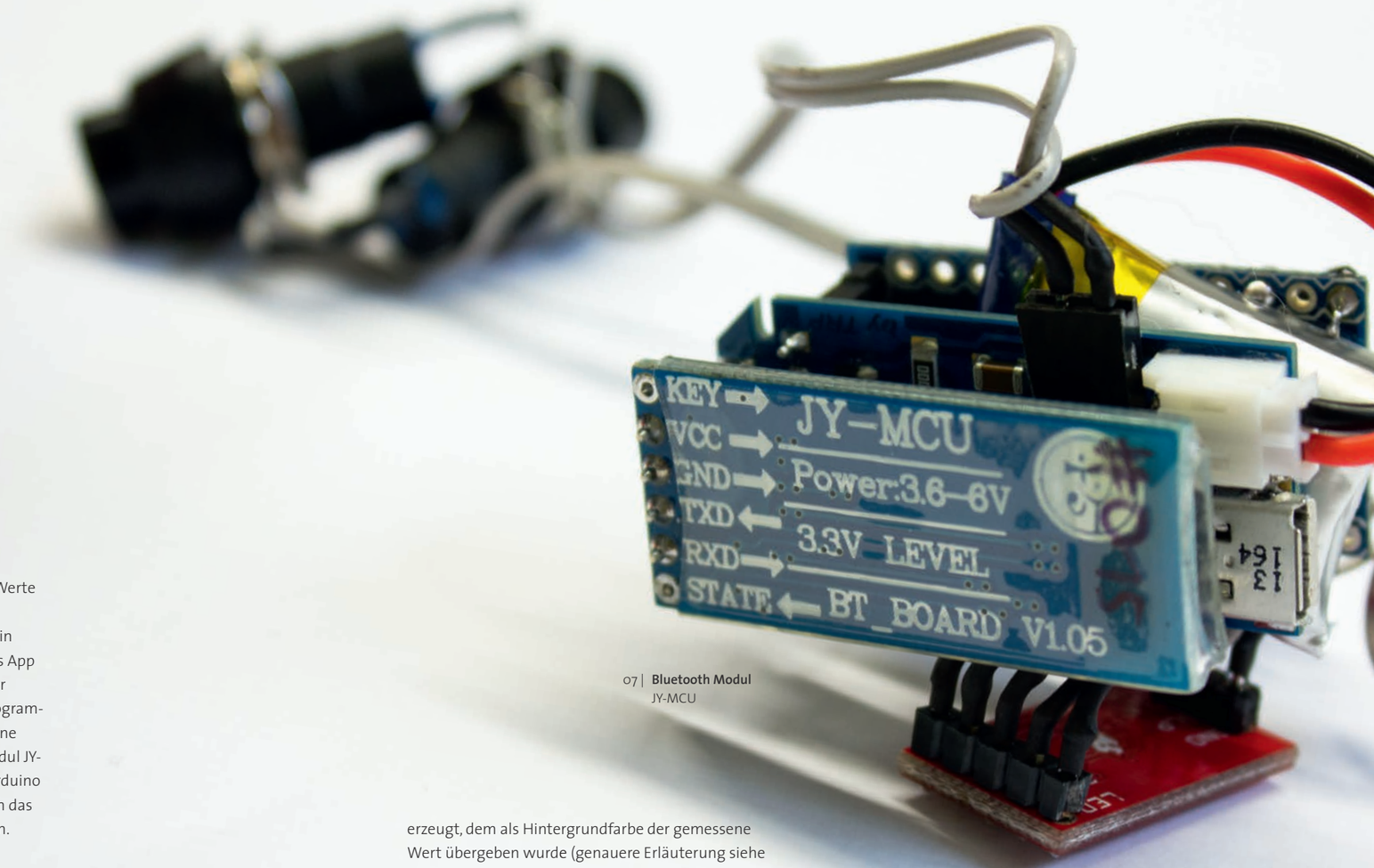
Farbsensor

Die eigentliche Umsetzung sollte mit Hilfe eines Arduinos realisiert werden - einer Elektronikplattform auf Basis einfach zu bedienender Hard- und Software. Die Hardware besteht aus einem einfachen I/O-Board mit einem Mikrocontroller und analogen und digitalen Ein- und Ausgängen.

Bereits zu Beginn wurde aufgrund der geringen Größe mit einem Arduino Mini gearbeitet. Mit Hilfe dessen sollte die oben geplante Technik umgesetzt werden. Hierzu wurde der Fokus zunächst auf die Hauptfunktion des Wearables gelenkt: eine Farbe ein zu scannen.

Dieses sollte mit Hilfe eines Farbsensors (Color Light Sensors) (Abb. 06) realisiert werden. Der Vorteil des Sensors besteht darin, dass dieser bereits über eine eingebaute LED verfügt, die für ein neutrales Licht sorgt. Um den Sensor richtig an den Arduino anzuschließen, wurden aus dem Internet diverse Schaltpläne untersucht und entsprechend nachgebaut. Ein wesentlicher Punkt, der beachtet werden musste war, dass der Sensor nicht mit mehr als 3,7 Volt versorgt werden durfte. Um dies zu gewährleisten, wurde ein Arduino mini mit 3,3 Volt genutzt, der über den VCC Output die 5 Volt, mit denen der Arduino über den Laptop versorgt wurde, in 3,3 Volt umwandelt. Für die richtige Datenübertragung mussten Verbindungsstücke an die Pins A4 und A5 gelötet werden, um diese mit den Pins SDA und SCL des Sensors zu verbinden, damit zwischen diesen eine I2C Kommunikation per Wire Library hergestellt werden konnte. Im Anschluss an die richtige Verkabelung und der Übertragung eines entsprechenden Beispielcodes aus dem Internet, konnten die ersten gemessenen Werte an die Konsole des Laptops übermittelt werden. Um zunächst die übertragenden Werte besser interpretieren zu können, wurde eine RGB LED angeschlossen, die über einen 100 und 150 Ohm Widerstand mit dem Arduino verbunden wurde. Um der RGB die Farben des Sensors zuzuweisen, wurden die entsprechenden Variablen an die RGB LED übergeben. Diese leuchtete anschließend in den vom Sensor gemessenen Farben.

06 | **Color Light Sensor**
ADJD-S311



07 | **Bluetooth Modul**
JY-MCU

Bluetooth Modul

Nachdem der Sensor wie gewünscht die Farbwerte gemessen und in der Konsole ausgegeben hatte, sollte in einem nächsten Schritt der Arduino mit dem Sensor, unabhängig vom Laptop, die Werte übermitteln. Dies war ein wesentlicher Schritt, da das Wearable auch später unabhängig vom Computer zum Einsatz kommen sollte. Um sich diesem Ziel zu nähern, wurde zunächst ein Bluetooth Modul (Modell JY-MCU) (Abb. 07) angeschlossen. Um dieses besser testen zu können, wurde eine einfache Print Ausgabe auf den Arduino programmiert. Um das bisherige „USB-to-Serial“ Modul, über welches der Arduino bisher mit dem Laptop verbunden war zu entnehmen, wurde eine 9 Volt Batterie angeschlossen. So konnte der Arduino unabhängig von dem Laptop mit Strom versorgt werden. Obwohl das Modul erfolgreich über den Laptop angesprochen werden konnte, funktionierte die Datenübertragung noch nicht und die Verbindung brach regelmäßig ab.

Um dieses Problem zu beheben, wurde die Datenübertragung nicht mehr, wie bisher über die Pins RXT und TXD vorgenommen, sondern über die digitalen Pins 10 und 11. In einem Code, der eine Bluetooth Verbindung mit dem Device erzeugt, wurden diese Pins zur Datenübertragung eingetragen. Die Datenübertragung konnte im Anschluss auf dem Laptop eingesehen werden. Nachdem diese Kommunikation erfolgreich hergestellt wurde, bestand der nächste Schritt darin, diese Verbindung mit einem Smartphone herzustellen und die Werte des Sensors an dieses zu übermitteln.
Hierzu wurde zunächst mit Hilfe von Eclipse ein entsprechendes Bluetooth Cordova Projekt als App auf das Smartphone übertragen. Nachdem der passende Bluetooth Code auf den Arduino programmiert wurde, konnte mit Hilfe der Chat-App eine Bluetooth Verbindung mit dem Bluetooth Modul JY-MCU hergestellt werden. Der zuvor auf den Arduino gespielte Code bewirkte, dass Beispielwerte an das Handy übermittelt und dort angezeigt wurden.

Die zuvor übermittelten Beispielwerte sollten nun durch die Werte vom Sensor ersetzt werden. Hierzu wurde zunächst lediglich ein gemessener Farbwert an Stelle der Beispielwerte übergeben (bluetooth.println(color.blue)). Wie gewünscht wurde der blaue Wert der gemessenen Farbe auf dem Smartphone angezeigt. Nachdem diese Übertragung problemlos funktionierte, wurden alle gemessenen Werte in einem String an das Smartphone übermittelt (bluetooth.println (string(color.red) + string(color.green) + string (color.blue));. Da die Farbe anders als erwartet und nicht wie bei RGB Farben üblich, in Zahlen von 0 – 255 dargestellt wurde, musste diese entsprechend umgerechnet werden. Hierzu wurden die insgesamt 1023 verschiedenen Werte, die der Sensor wiedergibt, durch 4 geteilt. Um die Darstellung der Farben zu prüfen, wurde mit Hilfe von HTML und JavaScript ein Div-Container im Chat-Programm erzeugt, dem als Hintergrundfarbe der gemessene Wert übergeben wurde (genauere Erläuterung siehe Seite 52, Kapitel Webtechnologien: Eingescannte RGB Werte zum Vergleich einbinden). Die Werte wurden im Anschluss an das Handy übermittelt und farblich in einem Div-Container angezeigt.

In einem weiteren Schritt sollten die beiden bisherigen Codes (zum Bluetooth-Modul und zum Farbsensor) zusammen geführt werden. Nicht das Zusammenfügen des Codes stellte sich an dieser Stelle problematisch heraus, sondern das gemeinsame Anschließen des Farbsensors und des Bluetooth Moduls. Nach intensiver Fehlerrecherche mit Hilfe des Multimeters wurde klar, dass die beiden Geräte mit zu wenig Strom versorgt wurden. Um diese Diagnose zu überprüfen, wurde ein Labornetzteil angeschlossen, mit dessen Hilfe die Stromversorgung erhöht wurde. Durch verschiedene Messungen mit dem Multimeter wurde klar, dass das Bluetooth Modul mindestens 3,6 Volt benötigt um funktionsfähig zu sein. Auf Grund des gemeinsamen Stromkreislaufes mit dem Farbsensor bekommt es durch den Arduino jedoch nur 2,7 Volt. Um dieses Problem zu beheben, wurde das Bluetooth Modul nicht mehr über den Arduino (3,3 Volt) mit Strom versorgt, sondern über den allgemeinen 5 Volt Stromkreis, über den auch der Arduino mit Strom versorgt wird. Diese Lösung sorgte dafür, dass alle Bauteile mit dem nötigen Strom versorgt wurden und einwandfrei liefen.

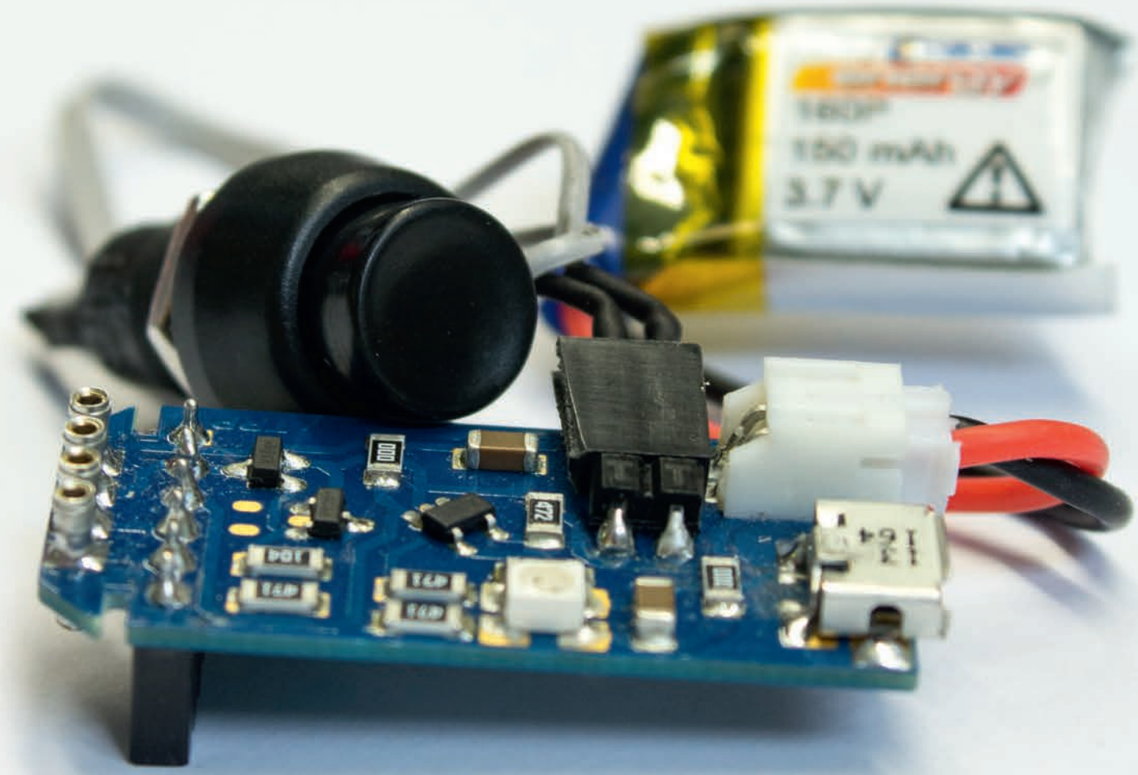
Stromversorgung

Da das spätere Wearable Device nicht über ein Labornetzteil mit Strom versorg werden sollte, galt es sich im Weitern Gedanken über eine andere Stromversorgung zu machen. Hierbei wurde auf einen Polymer Akku mit 3,7 Volt zurückgegriffen, der mit Hilfe eines USB Kabels jeder Zeit aufgeladen werden kann und zudem recht klein ist. Über diesen Akku wurden beide Module mit dem nötigen Strom versorgt. Jedoch war die Leuchtkraft des Farbsensors stark geschwächt. Scheinbar reichte die Stromzufuhr von 3,7 Volt nicht aus, um die LED komplett mit dem nötigen Strom zu versorgen. Da der Farbsensor mit 3,7 Volt betrieben werden durfte, wurde dieser mit in den allgemeinen Stromkreislauf, durch den nun 3,7 Volt fließen, eingebunden. So stand wieder genügend Strom für die LED zur Verfügung, die nun in ihrer vollen Intensität leuchtete.

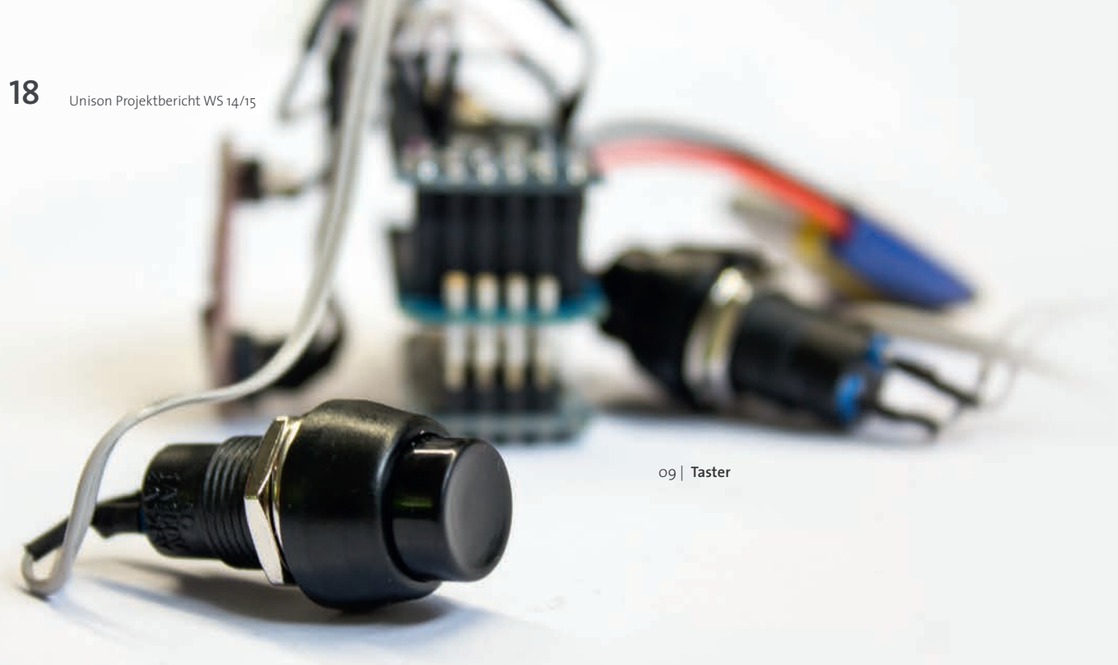
Schalter

Nachdem der Polymer Akku funktionsfähig eingebunden wurde, konnte der Fokus auf den Einsatz eines Schalters gelenkt werden. Mit Hilfe dessen sollte der Benutzer die Möglichkeit besitzen, das Wearable Device jederzeit an- oder ausschalten zu können. Bevor ein Schalte gekauft und eingebunden wurde, galt es sich Gedanken über die Art des Schalters zu machen. Hier gibt es verschiedene Ausführungen: Von einem Kipp-Schalter, einem Drückschalter, bis hin zu Schaltern, die lediglich eine Berührung benötigen um ein-/aus geschaltet zu werden. Da das hier konzipierte Wearable ein Gerät werden sollte, welches der Benutzer jederzeit in seiner Tasche tragen können soll, sollte auf Schalter verzichtet werden, die zu schnell und einfach angeschaltet werden können. Aus diesem Grunde fiel die Wahl auf einen Drückschalter, bei dem der Benutzer einen gewissen Druck ausüben muss, damit dieser einrastet.

Dieser Schalter wurde zwischen dem Plus Pol des Polymer Akkus geschaltet, wodurch nur Strom fließt, wenn dieser gedrückt wurde. Das An- und Ausschalten des Wearables konnte so problemlos realisiert werden.



08 | **Polymer Akku mit Schalter**



09 | **Taster**

Taster

Da der Sensor bisher ununterbrochen die gemessenen Werte an das Smartphone übermittelt, sollte ein Taster eingesetzt werden, der es ermöglicht, nur beim „Auslösen" einen gemessenen Wert an das Handy zu übermitteln und diesen dort anzuzeigen. Um den Taster in den bisherigen Arduino Aufbau zu integrieren, wurden zunächst verschiedene Schaltpläne aus dem Internet untersucht und entsprechend dem eigenen Aufbau angepasst. Mit Hilfe eines Multimeters wurde die Funktionsfähigkeit des Tasters überprüft. Bevor der Taster in Kombination mit dem Farbsensor eingesetzt werden sollte, wurde zunächst eine einfache LED angeschlossen, die durch drücken auf den Taster zum Leuchten gebracht werden sollte. Der entsprechende Code wurde auf den Arduino gespielt und sorgte für einen einwandfreien Testlauf, indem das Leuchten der LED durch den Taster ausgelöst wurde. Dieser Code konnte nun in den bereits vorhandenen Code integriert und wie folgt angepasst werden:

```
//wenn der Taster gedrückt wird
if (buttonState == Hight)
{
    // werden Werte über Bluetooth übersandt
    blutooth.println (colotr.red).....)
}
```

Vibrationsmotor

Damit der Benutzer im Anschluss an das Drücken des Tasters über das erfolgreiche Scannen der Farbe informiert wird, sollte ein Vibrationsmotor zum Einsatz kommen. Dieser soll dem Nutzer durch ein zweimaliges, leichtes Vibrationsfeedback verdeutlichen, dass die gemessenen Werte erfolgreich an das Smartphone übermittelt wurden. Hierzu wurde im Code ein Delay im Anschluss an das Übermitteln der Daten eingebunden, nach dessen Ablauf der Vibrationsmotor zwei Mal vibriert.

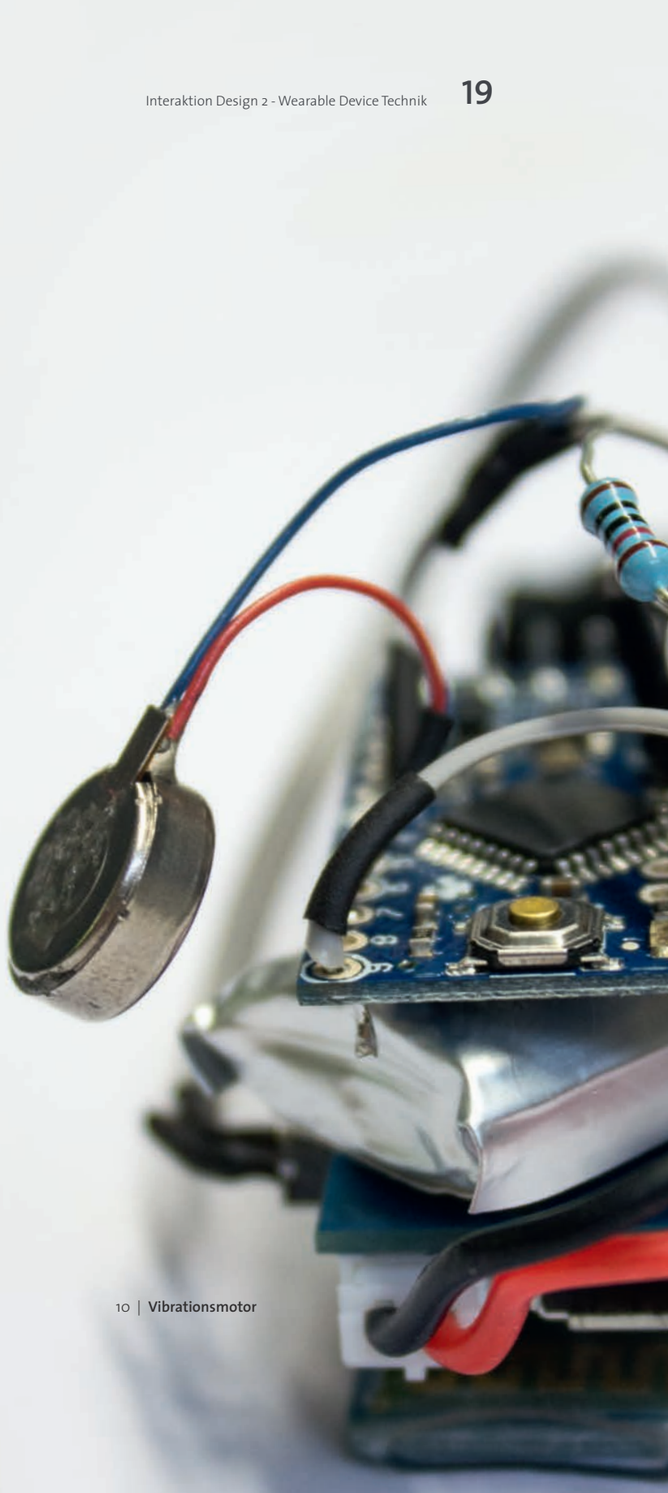
Da der Farbsensor nach dem Einschalten zunächst eine Kalibrierung durchführt, muss der Benutzer auch darüber informiert werden, wenn diese beendet ist. Erst dann kann er die gewünschten Farben verbindlich und neutral einscannen.

Um dies zu gewährleisten sollte abermals der Vibrationsmotor zum Einsatz kommen. Die Frage die sich zunächst stellte beschäftigte sich damit, woran erkannt werden kann, ob die Kalibrierung abgeschlossen ist. In dem zugehörigen Arduino Code konnte eine Libary identifiziert werden, die für die Kalibrierung verantwortlich ist. Der Vibrationsmotor sollte dementsprechend ein Feedback geben, wenn die Libary ausgeführt wurde. Um sicher zu gehen, dass diese wirklich komplett abgeschlossen ist, wurde ein Delay von 4 Millisekunden im Anschluss an den Kalibrierungs-Funktionsaufruf eingesetzt.

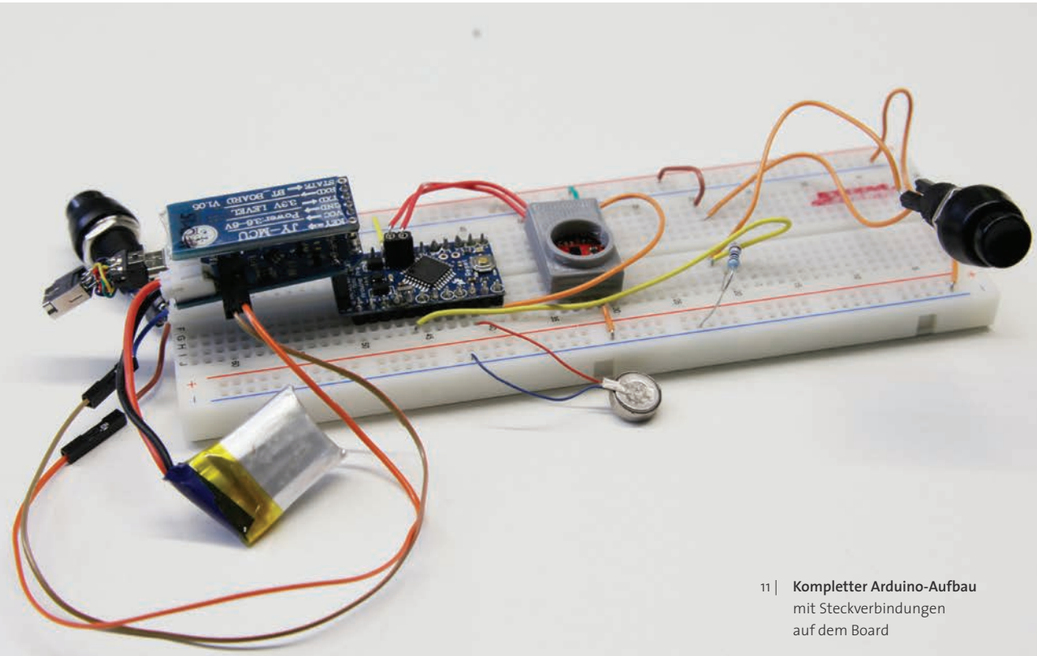
Um sicher zu gehen, dass diese wirklich komplett abgeschlossen ist, wird ein Delay von 4 Millisekunden im Anschluss an den Kalibrierungs-Funktionsaufruf eingesetzt.

```
// Kalibrierungs-Funktion wird aufgerufen
Colorsensor.calibrate();
 // es wird 4 Millisekunden gewartet
Delay(4)
//Vibrationsmotor wird gestartet
digitalWrite(motorpin, HIGH);
//Der Motor läuft eine halbe Sekunde
delay (500),
//Vibrationsmotor wird abgeschaltet
digitalWrite(motorPin, low)
```

Im Anschluss an die Einbindung des Codes und die Integrierung des Vibrationsmotors, wurde an den gewünschten Stellen das Vibrationsfeedback realisiert.



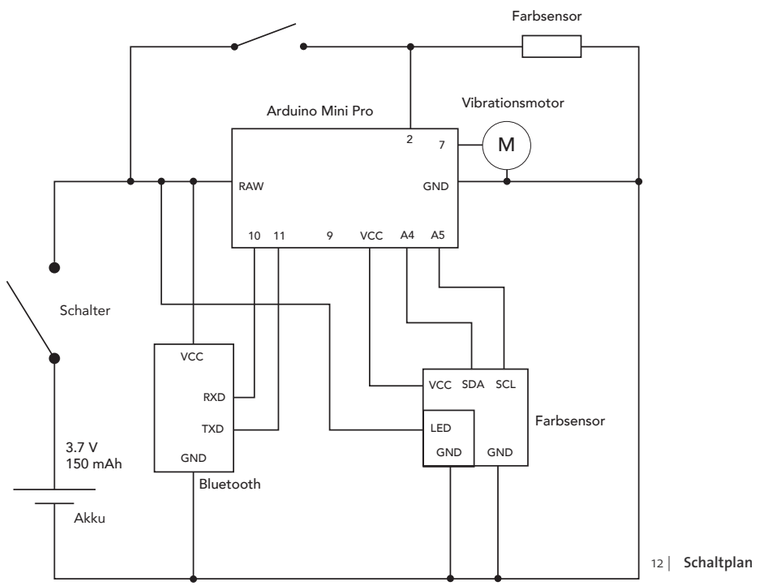
10 | **Vibrationsmotor**



11 | **Kompletter Arduino-Aufbau** mit Steckverbindungen auf dem Board

13 | **Löten der Litzen** an den Tasters

14 | **Kompletter Arduino-Aufbau** zusammengelötet mit Schrumpfschläuchen



12 | **Schaltplan**

Löten

Nachdem alle nötigen Bestandteile für das Wearable Device erfolgreich mit einander kombiniert und getestet wurden, galt es in einem letzten Schritt den Aufbau von dem Board zu nehmen und die einzelnen Steckverbindungen fest miteinander zu verlöten. Hierzu wurden Litzen genutzt, welche auf Grund ihrer Flexibilität gut für eine kompakte Zusammenführung der Bauelemente geeignet sind. Diese wurden entsprechend zurechtgeschnitten und an die genutzten Verbindungsstellen gelötet. An den Ground und an den VCC Ausgang des Arduinos wurde zunächst jeweils eine Litze befestigt. Diese wurde im Anschluss dazu genutzt, alle Minus- bzw. Plus Verbindungen zusammenzufügen und die Bauteile über einen Plus/Minus-Pol mit dem nötigen Strom zu versorgen.

Damit die Elektronik in dem späteren Gehäuse vor Kurzschlüssen geschützt ist, wurden die einzelnen Verbindungen mit Hilfe von Schrumpfschlauchen isoliert. Diese wurden jeweils an den Löt-Verbindungen eingesetzt und durch Föhnen befestigt.

Auf diese Art und Weise wurden alle Bauteile, unabhängig vom Board, so kompakt wie möglich miteinander verlötet und für einen sicheren und problemlosen Einsatz im Wearable aufbereitet.

Der technische Teil der Umsetzung war somit erfolgreich abgeschlossen. In einem nächsten Schritt konnte der Fokus auf das Gehäuse und dessen äußeres Erscheinungsbild gelenkt werden.

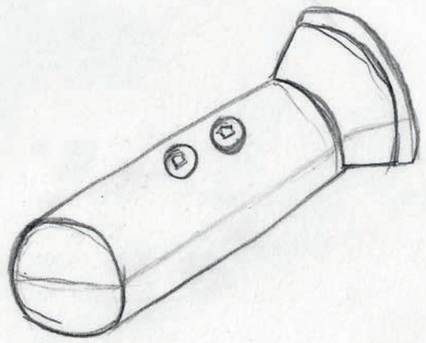
WEARABLE DEVICE
GEHÄUSE

Da an dieser Stelle bereits fest stand, welche Bauteile für die technische Realisierung verwendet werden mussten und diese bereits fest miteinander verbunden waren, konnten nun erste Skizzen für das Case angefertigt werden. Bei der Erstellung der Skizzen musste einiges bedacht werden: Hierzu zählte zum Beispiel, dass die Knöpfe gut für den Benutzer erreichbar sein müssen oder dass der Farbsensor, bevor er zum Einsatz kommt, mit einer weißen Fläche zur Kalibrierung abgedeckt sein muss. Außerdem war es wichtig, dass der Prototyp jederzeit auseinander und wieder zusammengesteckt werden kann, sodass die Technik entnommen und geändert werden kann.

Gerade die Verdeckung des Farbsensors stellte sich als Herausforderung dar. Es musste ein Mechanismus gefunden werden, der leicht zu öffnen und zu schließen ist. Außerdem musste der Mechanismus für den Prototypen umsetzbar sein und es sollten möglichst zusätzliche Teile, wie zum Beispiel eine Verdeck-Kappe, vermieden werden, damit man diese nicht beim Einscannen zusätzlich festhalten muss.

15 | **Wearable Device**
die fertige Version

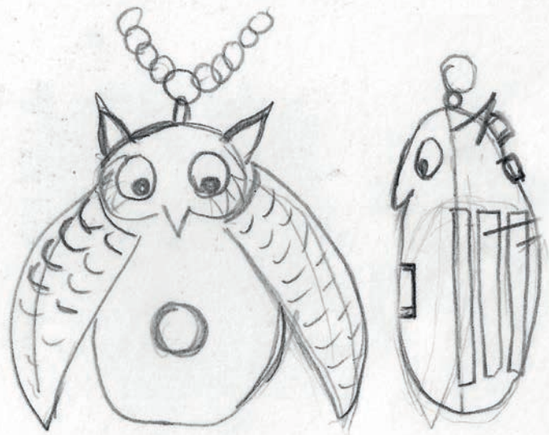
16 | **Entwurf 1**
in Form einer Taschenlampe



17 | **Entwurf 2**
„Farbschnüffler“

18 | **Entwurf 3**
Kleid

19 | **Entwurf 4**
Eule



Entwurf 1

Der erste Entwurf erinnert in seiner Form an eine Taschenlampe. Durch die runde zylindrische Form des Case liegt dies gut in der Hand und auch die Knöpfe sind in einer angenehmen Handhaltung erreichbar. Der Farbsensor liegt hinter der trichterförmigen Abschirmung.

Bei näherer Betrachtung dieses Entwurfes stellte sich jedoch heraus, dass die Abschirmung viel zu groß bedacht wurde. Die Öffnung für den Sensor muss möglichst klein sein, damit man den zu prüfenden Stoff auch 2-3 mm vor den Sensor halten kann.

Entwurf 2

Der zweite Entwurf ergab sich aus der Form der Hand bei einer natürlichen Greifbewegung. Die Finger umschließen dabei in einer Wellenform das Objekt, so dass die Hand besonders bequem das Objekt umfassen kann. Aus der Kombination dieser Wellen, der Trichterform für den Sensor und dem Arduino selbst, ergab sich dann eine Form, die einem Tier mit vier Beinen sehr ähnlich sah. Im Bauch des Tieres würde die Arduino, Bluetooth & Akku Kombination stecken. Die beiden Schalter wären an der Position der Ohren befestigt und der Sensor würde die Nase des Tieres bilden. Im Rahmen dieses Entwurfes kam auch die Idee auf, die App „Farbschnüffler“ zu nennen, da das Tier so zu sagen an den Farben schnüffelt, um sie einzuscannen. Beim zweiten Blick auf diesen Entwurf erschien dieser jedoch zu kindlich und nicht passend für die zuvor definierte Zielgruppe.

Entwurf 3

Der dritte Entwurf ergab sich aus der Form der Bauteile. Dieser Ansatz orientierte sich an dem Leitsatz „form follows function“. Die Kombination aus den Bauteilen übereinander gestapelt, so wie es in dem Prototyp später auch sein sollte, ergab einen menschlichen Korpus. Da es bei dem gesamt Konzept um Kleidungsstücke und Mode geht, wurde dieser Korpus leicht verändert erst zu einer Schneiderpuppe, später dann zu einem Kleid geformt. Dies liegt durch die längliche Form ebenfalls gut in der Hand und spricht optisch auch die Zielgruppe an. Jedoch stellte sich die Frage, woran man dieses Kleid befestigt, ohne dass es störend wirkt. Der Nutzer soll nicht das Gefühl bekommen, ständig ein zusätzliches Gerät mit sich herum tragen zu müssen. Außerdem gab es keine gute Lösung den Sensor für die Kalibration zu verdecken, ohne dass der Nutzer eine Verdeck-Kappe mit sich herum tragen müsste.

Entwurf 4

Der finale Entwurf stellt eine Eule dar, welche die ganze Technik in sich versteckt. Die Eule sollte als hübsches Schmuckstück um den Hals getragen werden können. Der Entwurf ist von einem Amulett inspiriert, welches man aufklappen kann. Für die Optik wurde eine Eule gewählt, da dieses Motiv momentan besonders beliebt ist und auch gerne als Schmuckstück getragen wird. Somit könnte unsere Zielgruppe eine kleine silberne oder goldene Eule immer mit sich herumtragen. Dieses Motiv bot sich vor allem dadurch an, dass der Farbsensor durch eine raffinierte Technik verdeckt werden konnte. Wenn die Flügel geschlossen sind, ist der Sensor verdeckt und kann sich an der weißen Innenseite der Flügel kalibrieren. Wenn der Sensor benötigt wird, können die Flügel einfach über die Ohren aufgeklappt werden. Dies hat den Vorteil, dass es keine Abdeckkappe gibt, die man verlieren kann.

BAU DES PROTOTYPEN

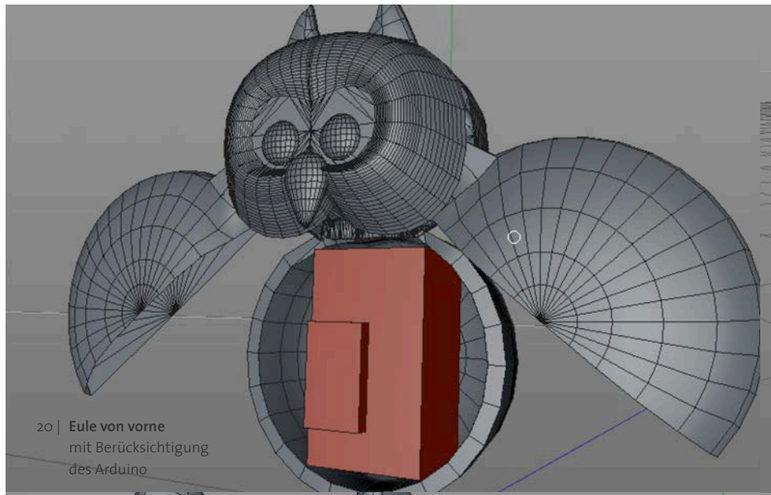
Für den späteren 3D Druck des Prototypen wurde ein 3D Objekt in Cinema 4D erstellt. Dabei musste beachtet werden, dass der Innenraum der Eule auch im späteren Druck ein Hohlraum ist und der Raum nicht gefüllt werden darf.

Zuerst wurde der Aufbau der Eule mit Grundelementen geplant. Dabei war es wichtig darauf zu achten, dass sich die Flügel frei bewegen können müssen. Außerdem musste die Eule aus mehreren Einzelteilen bestehen, damit sie sich für spätere Wartungsarbeiten öffnen und schließen lässt. Bei diesem ersten Test-Modell aus Grundelementen in Cinema 4D fielen Konstruktionsfehler auf, die im späteren Endentwurf mit Bedacht wurden. So mussten die Flügel zum Beispiel zwischen Gesicht und Hinterkopf gesteckt werden und nicht wie erst geplant, direkt am Gesicht befestigt werden.

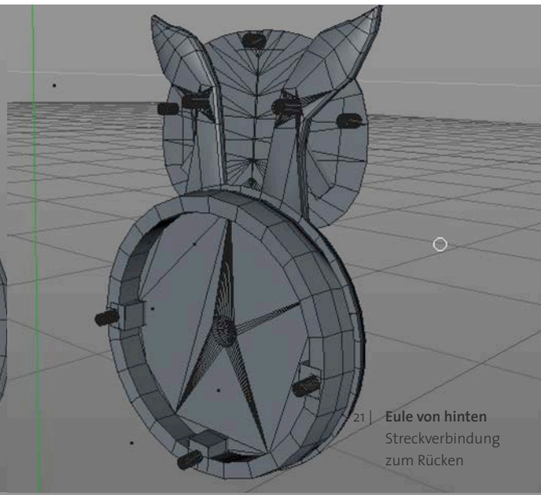
Schließlich wurde der komplette Arduino Aufbau Millimeter genau gemessen, sodass diese Maße die Grundlage für die gesamte Eulen-Konstruktion waren. Der Bauch der Eule besteht aus einer großen Halbkugel, in welcher der Arduino-Aufbau verschwindet. Nur eine kleine runde Öffnung an der Bauchseite wurde offen gelassen, hinter der sich später der Farbsensor befindet. Die Öffnung wurde mit Bedacht sehr klein gewählt, damit die Abschirmung möglichst groß ist. Da man die zu messende Farbe möglichst 1-3mm vor den Sensor halten soll, konnte keine große Abschirmung angelegt werden und die Bauchwand musste besonders dünn sein.

Die Flügel, welche den Sensor im geschlossenen Zustand verdecken sollten, wurden aus einem geschwungenen Pfad erstellt. Dieser wurde schließlich extrudiert, damit ein räumlicher Körper entsteht. Anschließend wurden noch Feinheiten wie Rundungen oder spitze Ohren ausgearbeitet.
Die Abdeckung des Sensors wurde durch eine runde Platte an einem der Flügel sichergestellt. Diese liegt genau über dem Sensor, wenn die Flügel geschlossen sind. So kann gewährleistet werden, dass der Sensor vollständig abgedeckt ist und sich nicht genau an der Schnittstelle der beiden Flügel befindet. Außerdem wurden Löcher im Ankerpunkt der Flügel angelegt. Um diese sollten sich die Flügel später drehen.

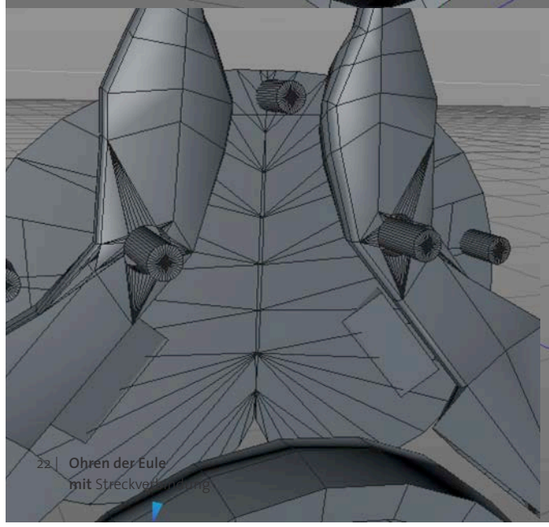
Die Gegenstücke zu den Löchern der Flügel sind kleine Stäbchen, auf welche die Flügel gesteckt werden. Diese wurden am Gesicht der Eule angebracht. Damit die Flügel im aufgeklappten Zustand mehr Halt haben und nicht sofort wieder zuklappen, wurden kleine Stützen an der Rückseite des Gesichts entwickelt, auf denen die Flügel liegen können. Das Gesicht der Eule wurde mit mehreren Pfaden ausgearbeitet, die schließlich für das Loft Tool zu einer Fläche verbunden wurden. Dadurch war es möglich der Eule ein individuelles Gesicht zu geben.



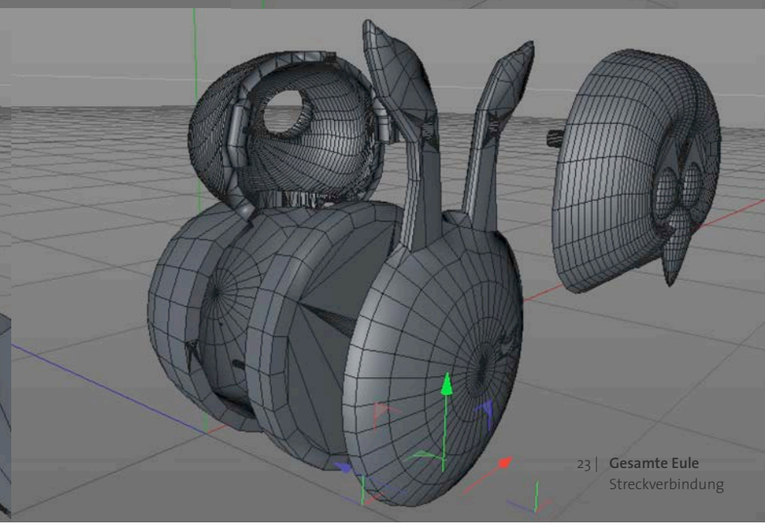
20 | **Eule von vorne** mit Berücksichtigung des Arduino



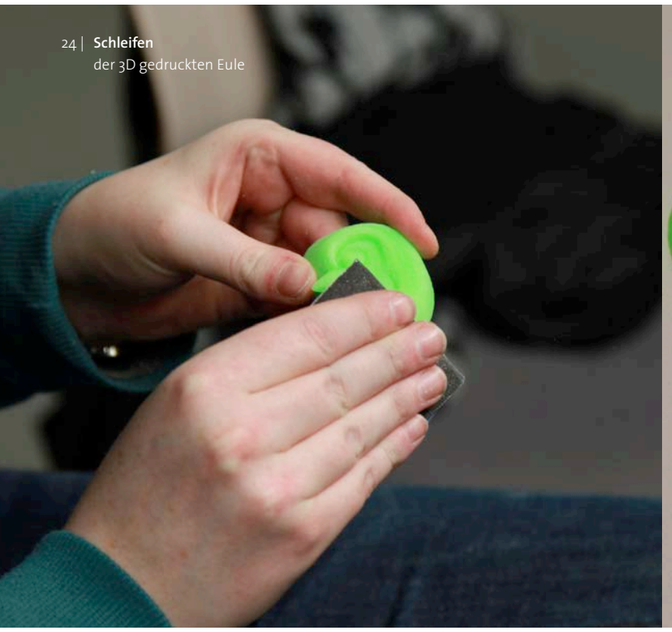
21 | **Eule von hinten** Streckverbindung zum Rücken



22 | **Ohren der Eule mit** Streckverbindung

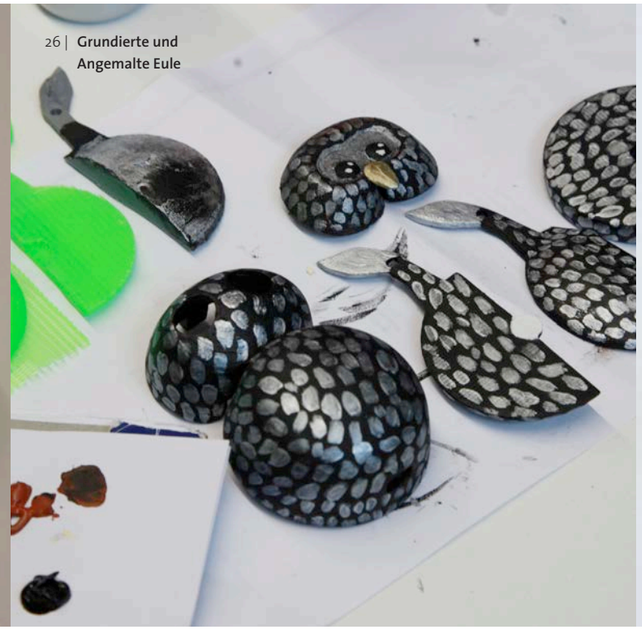


23 | **Gesamte Eule** Streckverbindung



24 | **Schleifen**
der 3D gedruckten Eule

25 | **Kompletter 3D**
geschliffen



26 | **Grundierte und Angemalte Eule**

27 | **Innenleben**
mit Technik

Des Weiteren mussten bei der Erstellung des Prototypen Öffnungen für die Knöpfe und die Ladestation berücksichtigt werden. Auch diese Elemente wurden zuvor möglichst genau ausgemessen und anschließend ebenfalls in der Cinema 4D Datei angelegt. Durch ein Boolean Objekt konnten diese Elemente von der eigentlichen Mesh der Eule abgezogen werden, sodass eine passgenaue Öffnung für die Knöpfe und für die Ladestation entstand.

Die einzelnen Elemente sollten über eine Steckverbindung zusammengefügt werden können, damit man im Notfall Wartungsarbeiten an der Technik durchführen kann. Bei einem kurzen Testdruck dieser Steckverbindung stellte sich jedoch heraus, dass der 3D-Drucker die Nieten viel zu ungenau druckt, sodass es nicht möglich war, diese in die vorgesehenen Löcher zu stecken. Auch wurden die Nieten vom 3D Drucker nur auf die Flächen aufgesetzt und hatten somit keinen festen Halt auf dem Element. Dies führte dazu, dass die Nieten sehr leicht abbrachen.

Die alternative Lösung bestand darin, die Nieten durch kleine Metallnägel zu ersetzen. Zuerst wurden kleine Löcher mit einen Handballenbohrer in das Gehäuse gebohrt und den kleinen Nägeln der Kopf abgeknipst. Die so entstandenen Metalldübel wurden in die gebohrten Löcher gesteckt und dienten somit als Verbindungsstück zwischen den Einzelteilen. Diese ließen sich nun aufeinander stecken.

Nachdem die Eule komplett gedruckt war und alle Teile perfekt aufeinander passten, wurde mit dem Schleifen der Einzelteile begonnen, damit die Oberfläche möglichst glatt erscheint. Vor allem die Rundungen wurden so glatt geschliffen, dass die einzelnen Druckringe nicht mehr zu erkennen waren. Kleine Feinheiten, wie das Gesicht der Eule, waren jedoch schwierig zu glätten, da man diese Stellen nur sehr schlecht mit dem Schleifpapier erreichte.

Da die Eule aus knallgrünem Filament gedruckt wurde, sollte sie einen etwas neutraleren Anstrich bekommen. Zuerst wurde jedes Einzelteil mit schwarzer, deckender Acrylfarbe grundiert. Anschließend wurde mit einzelnen Pinselstrichen ein Muster auf die Eule gemalt, welches die Federstruktur widerspiegeln sollte. Für die Bemalung wurde goldene und silberne Farbe benutzt, sodass die Eule ein hochwertigeres Aussehen erhält. Insgesamt sollte die Eule durch die Bemalung wie ein Schmuckstück aussehen, welches die Zielgruppe gerne mit sich trägt.

Ausblick

Da es sich bei dieser ersten Konstruktion der Eule um einen Prototyp handelt, sind an vielen Ecken noch Verbesserungen möglich. So wäre es zum Beispiel sinnvoll, dass die Flügel beim Aufklappen einrasten, so dass sie offen stehen bleiben. So ein kleinteiliger Mechanismus ließ sich in dieser Version des Prototyps jedoch noch nicht realisieren. Auch würde die Endversion der Eule viel kleiner ausfallen, da sich die Technik, die sich im Bauch der Eule befindet, noch verkleinern würde.

RESÜMEE

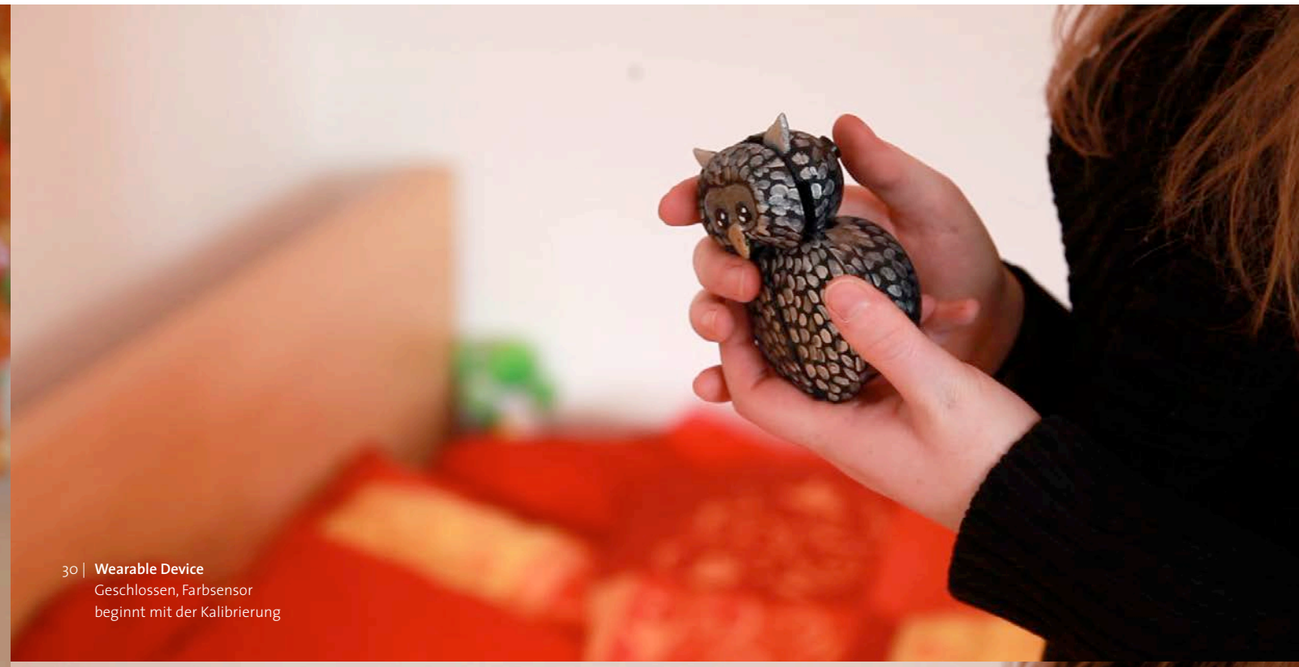
Die gesamte Realisierung des hier entwickelten Prototypen forderte den Bereich der Elektrotechnik und die Funktionsweise der Bauteile zu verstehen. Angefangen von dem Ansteuern einer einfachen LED, wurden die Fähigkeiten im Laufe des Projekts schrittweise erweitert, bis sie zur Schaffung des hier realisierten Prototypen führten.

Im Laufe des Projekts wurde vermehrt das Multimeter eingesetzt, um den Stromfluss zu prüfen und Spannungen zu messen. Mit dessen Hilfe konnten viele Probleme gefunden und selbstständig behoben werden. Gemeinsam wurde bei Schwierigkeiten recherchiert und eine Lösung erarbeitet, die zum gewünschten Ziel beitrugen. Nach und nach entwickelte sich ein fundiertes Wissen im Bereich der Elektrotechnik und in dem Zusammenspiel der einzelnen Bauteile. Die Ursache von Fehlern konnten zum Ende des Projektes immer schneller identifiziert und gelöst werden.

Insgesamt konnte gemeinsam im Team das geplante Wearable umgesetzt werden. Hierbei wurde nicht nur die funktionsfähige Technik realisiert, sondern auch ein Gehäuse gebaut, welches effizient die Technik in sich vereint und dem Nutzer die Möglichkeit bietet, mit einem ausgearbeiteten Mechanismus Farben einzuscannen. Das hier erschaffene Schmuckstück in Form einer Eule wurde nicht nur optisch ansprechend gestaltet, sondern besitzt auch die Nötige Funktion, die für das Projekt erforderlich ist.

28 | **Wearable Device**
in Form einer Eule

29 | **Wearable Device**
Unison



30 | **Wearable Device**
Geschlossen, Farbsensor
beginnt mit der Kalibrierung

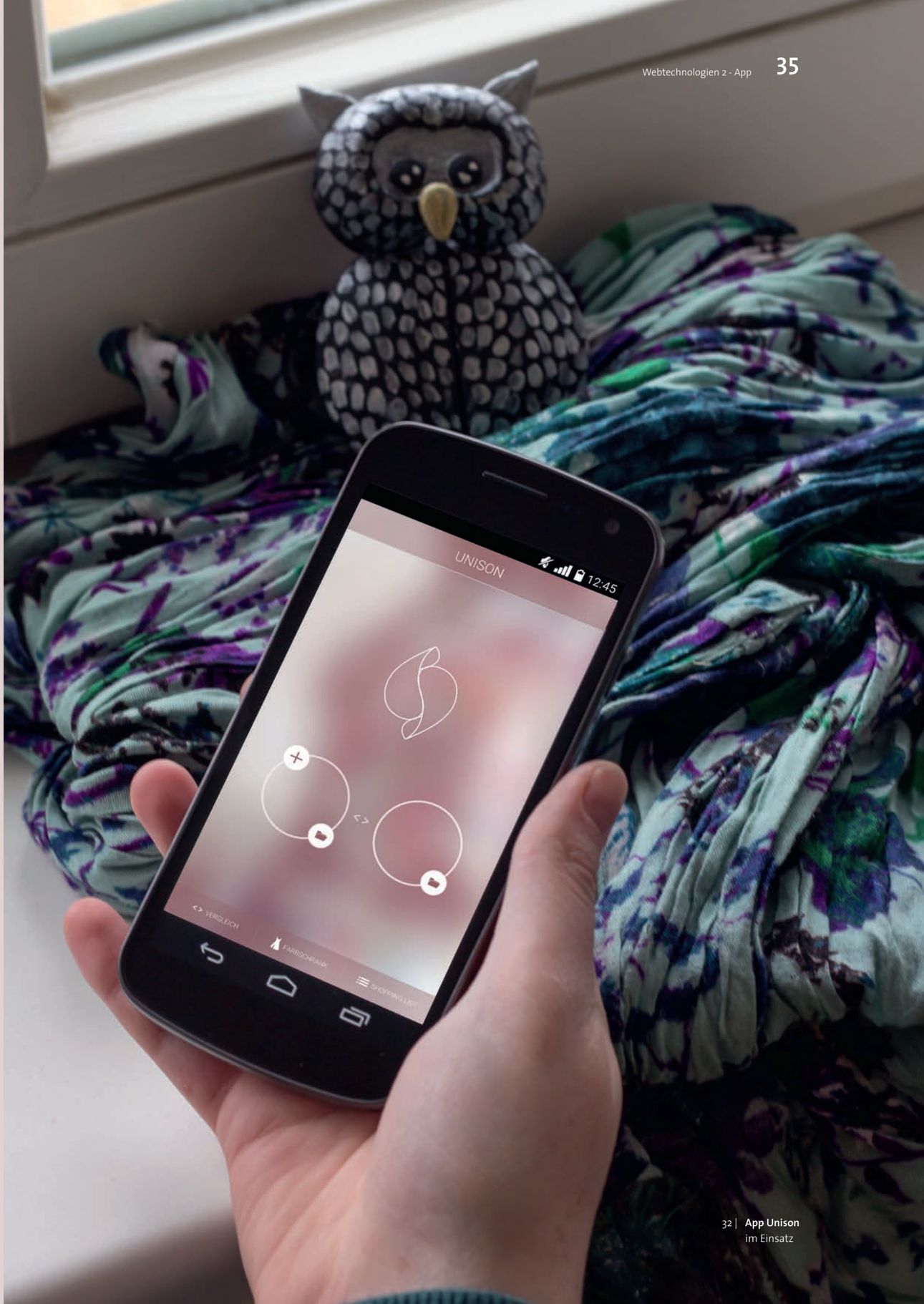
31 | **Wearable Device**
Geöffnet um Farben
einzuscannen

WEBTECH 2

Im Rahmen des Moduls „Webtechnologien 2“ ging es im dritten Semester hauptsächlich um die Erstellung einer Web App mit HTML-/CSS-/Java-Script unter Verwendung von Cordova (Framework zur Erstellung von nativen Apps) und den richtigen Einsatz spezieller Frameworks wie z.B. jQuery Mobile (Web-Framework), Sencha Touch (Framework zur Entwicklung von Apps für mobile Endgeräte oder Browsern) oder Laravel (PHP-Framework).

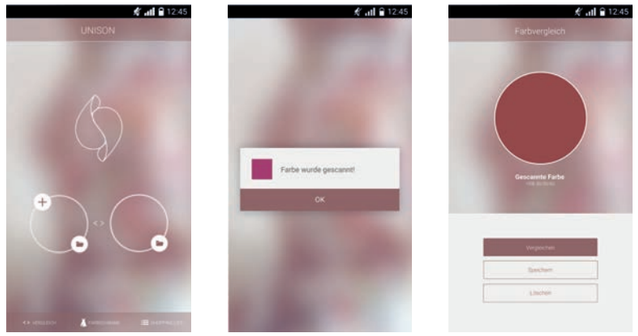
Die hierzu erlernten Grundlagen sollten nun dazu dienen für das oben aufgeführte Konzept Unison eine zugehörige native App mit Cordova auf Basis einer HTML/CSS/JS-App zu realisieren.

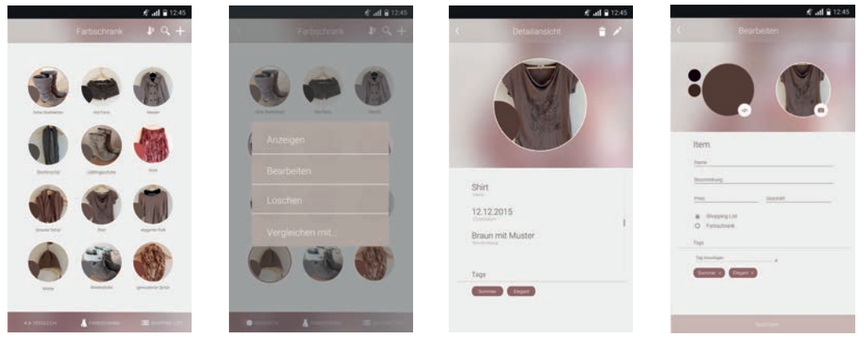
Im Folgenden wird zunächst eine detaillierte Beschreibung des Konzepts und des Aufbaus der App gegeben, bevor im Anschluss genauer auf die Umsetzung eingegangen wird. Hierbei wird explizit auf einzelne Funktionen, bestimmte Probleme und Erkenntnisse eingegangen.



32 | **App Unison** im Einsatz

APP
KONZEPT





Bereits zu Beginn war es abzusehen, dass die App von ihrer Funktionsweise recht umfangreich werden könnte. Für die Konzeption sollten zunächst keine Einschränkungen vorgenommen werden. Auch Funktionen, die bei der Umsetzung der App nicht realisiert werden, sollen an dieser Stelle Berücksichtigung finden und die App in ihrer optimalen Ausführung widerspiegeln.

Der Ablauf und die einzelnen Funktionsweisen der App wurden zunächst mit Hilfe eines simplen Mock-Ups definiert, erarbeitet, visualisiert und schrittweise verbessert. Grob lässt sich die App in vier Hauptbereiche gliedern, den Farbschrank, die Shopping List, das Verarbeiten einer eingescannten Farbe und das Vergleichen von Items.

Der „Farbschrank“ ist eine Art virtuelles Abbild des eigenen realen Kleiderschranks. Hier können alle Kleiderstücke als Item mit einem entsprechendem Foto, der Farbe (in RGB), einem Beschreibungstext und selbstdefinierte Tags (z.B, Sommer, elegant) gespeichert werden. Die Tags sollen eine spätere Suche im virtuellen Kleiderschrank erleichtern und auch zur Sortierung genutzt werden können. Mit Hilfe einer Detailansicht, die durch einen Tap auf ein Item aufgerufen wird, erhält der Nutzer eine genaue Übersicht über das ausgewählte Item. Hier kann dieses auch bearbeitet oder angepasst werden, indem z.B. ein neues Foto aufgenommen wird, oder falls keins vorhanden war, eins hinzugefügt werden.

Zusätzlich zum Farbschrank verfügt die App über eine „Shopping List“, in der die Kleidungsstücke beim Einkaufen gespeichert werden können, wenn man diese später noch einmal einsehen, aber zur Zeit nicht kaufen möchte. Auch hier können die Items abermals durch einen Tap in einer Detailansicht angezeigt und bearbeitet werden. Grundsätzlich wird die Shopping List als eine Liste angezeigt, die nach Datum sortiert werden soll, so dass der Nutzer unmittelbar das zuletzt eingescannte Item vor Augen hat. Im optimalen Ansatz soll der Benutzer die Anzeige aber zwischen einer Listen- und einer Kachelansicht wechseln können.

Wird eine Farbe mit Hilfe des zugehörigen Wearable Device eingescannt, erscheint in der App ein neuer Screen, in dem der Nutzer die Möglichkeit hat, die eingescannte Farbe zu löschen, zu speichern oder diese direkt mit einem, mehreren oder allen Item(s) aus dem Farbschrank und aus der Shopping List zu vergleichen. Im Anschluss erhält der Nutzer eine detaillierte Auflistung über die Vergleichsergebnisse in Prozent. In dieser Ansicht besteht die Möglichkeit, die zuvor eingescannte Farbe entweder in der Shopping List oder im Farbschrank, falls man dieses Item gekauft hat, noch einmal zu speichern oder diese wieder zu verwerfen, wenn man diese nicht mehr für spätere Vergleiche benötigt.

Der Vergleich kann nicht nur mit Hilfe eines neu eingescannten Items durchgeführt werden, sondern auch durch die Startseite der App. Hier kann der Nutzer eine Farbe neu einscannen oder eine vorhandene Farbe, aus dem Farbschrank oder der Shopping List, mit einem oder mehreren gespeicherten Items vergleichen.

Im Anschluss an die optimale Konzipierung der App wurde diese hinsichtlich relevanter Funktionen untersucht, die für den zu erstellenden Prototypen unerlässlich sind. Hierzu zählte vor allem die Anzeige einer Shopping List und eines Farbschrankes, in dem der Nutzer die Möglichkeit hat, sich die entsprechenden Items anzuzeigen. Auch das Hinzufügen eines neuen Items in dem Farbschrank sollte realisiert werden. Des Weiteren wurde der Vergleichsfunktion eine hohe Priorität zugeschrieben, da diese die Hauptfunktion der App darstellt. Somit soll in dem Prototyp der App die Farbe, die mit dem Wearable eingescannt wurde, angezeigt werden und gespeichert, gelöscht oder mit gespeicherten Items verglichen werden können. Das Ergebnis des Vergleiches soll auf einer separaten Seite in Prozent, absteigend dargestellt werden.

Funktionen, wie das Sortieren oder das Ändern der Ansicht, werden in diesen Prototypen nicht berücksichtigt, da ihnen eine untergeordnete Rolle zugeschrieben wurde.

APP
UMSETZUNG

2.1 Layoutumsetzung mit JQuery Mobile

Bevor mit der Programmierung der wichtigen Hauptfunktionen begonnen werden konnte, galt es zunächst das zuvor entwickelte Layout mit Hilfe von HTML und CSS „nachzubauen“. Bei der Umsetzung wurde auf das Touchscreen optimierte Web Framework jQuery Mobile zurückgegriffen, mit dessen Hilfe eine einfache und schnelle Realisierung des geplanten App Layouts gewährleistet wurde. JQuery Mobile ist ein auf HTML5 basierendes User Interface System, welches bereits einige nützliche UI Elemente beinhaltet, die so gestylt sind, dass sie optimal auf Geräten mit Touchscreen genutzt werden können. Eine individuelle Anpassung dieser Elemente kann jederzeit mit CSS Befehlen vorgenommen werden.

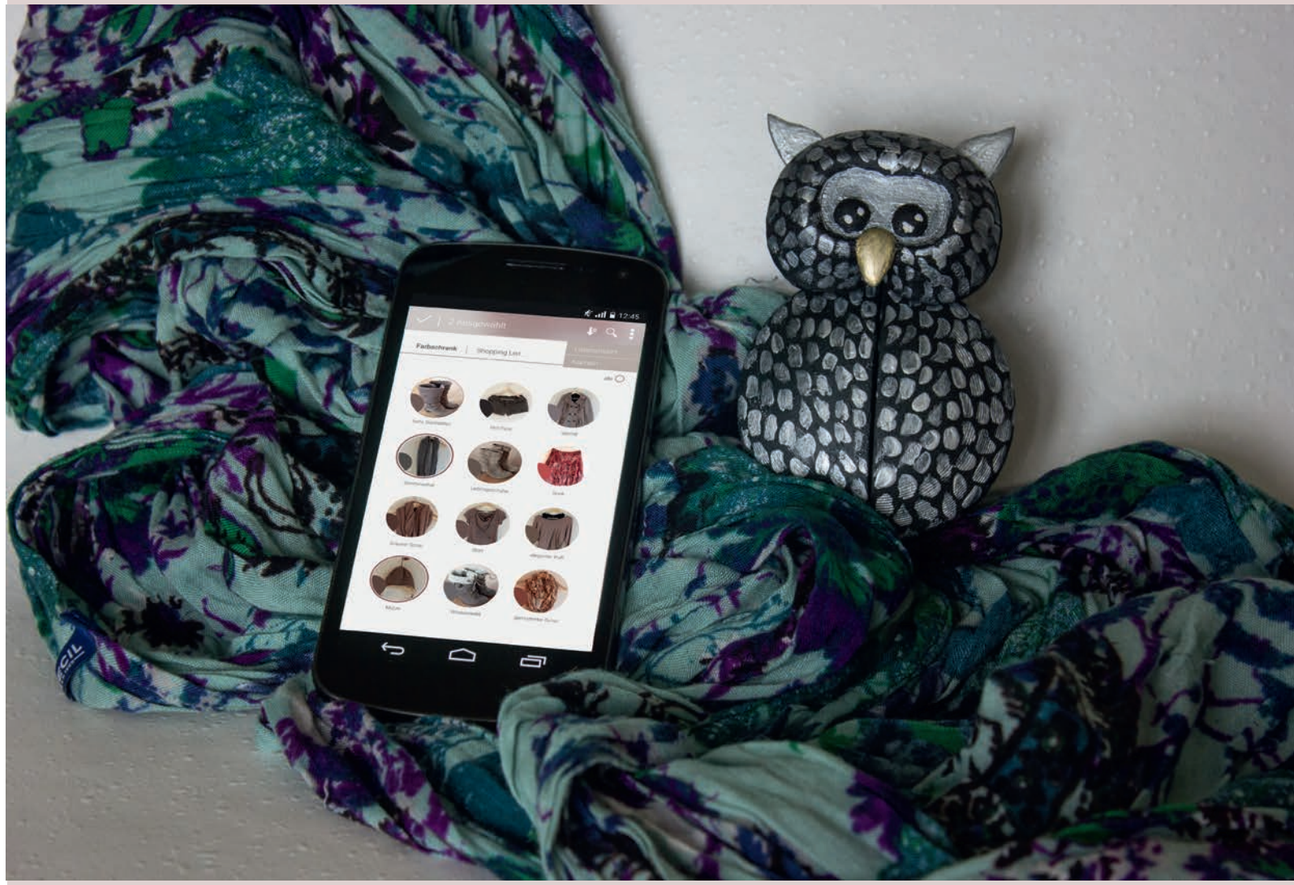
Nachdem das Framework im head Bereich der index. html Datei entsprechend eingebunden wurde, konnte die Struktur der geplanten App mit Hilfe der vordefinierten jQuery Mobile-Elemente nachgebaut werden.

Die einzelnen Seiten der geplanten App wurden mit Hilfe des Page-Elements erzeugt, wobei jede Seite durch das custom Attribut data-title einen eigenen Titel besitzt. Durch eine selbstdefinierte ID können die Seiten an einer anderen Stelle über das <a>-Tag problemlos aufgerufen werden. Hierbei erfolgt automatisch von jQuery Mobile eine Transition, die individuell in Typ und Richtung angepasst werden kann.

Die mit Hilfe des Page-Element erzeugten Seiten wurden unter Bezugnahme weiterer Elemente in Head (data-role=“header“), Content (class=“ui-content“) und Footer (data-role=“footer“) unterteilt. Der Head-Bereich besteht größtenteils aus einer <h1> </h1>, welche die Überschrift der Seite anzeigt und aus entsprechenden Icons. Bei der Anzeige der Icons wurde der Stil von jQuery Mobile überschrieben und ein eigens definierter Style (icon-style) definiert, mit dessen Hilfe die Icons wie geplant, ohne die von jQuery Mobile vordefinierten Button Eigenschaften in der App platziert werden konnten.

Der eigentliche Content Bereich unterscheidet sich je nach Seite. Verschiedene Seiten aus der App sollen in ihrer späteren Funktion automatisch aus den selbst gespeicherten Daten erzeugt werden. Für diese Seiten wurde ein Template zunächst statisch mit Hilfe von CSS und HTML erstellt und eingebunden. Durch den Einsatz bestimmter CSS- Eigenschaften und der Verwendung von SVG-Dateien, konnten die Ansichten wie in der geplanten App umgesetzt werden. Insgesamt wurde auf Button, Formular und Listenelemente zur Erstellung der Web App zurückgegriffen.

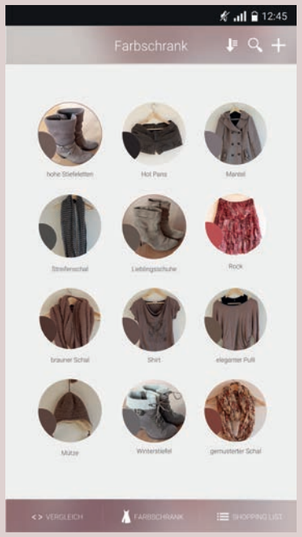
Der Footer enthält lediglich die dreiteilige Navigation, die durch eine Liste in dem Costum Attribut <data-role=“Navbar“> erzeugt werden konnte.



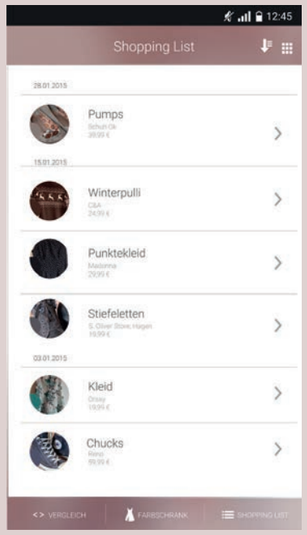
33 | **Layoutumsetzung**
Items zum Vergleichen auswählen

2.2 JSON Datei einbinden und anzeigen

Nachdem die App in Layout und Aufbau realisiert wurde, galt es in einem nächsten Schritt die bisher fest definierten Inhalte in der intex.html durch einen externen Datensatz dynamisch zu erzeugen. Dies war sowohl für die Anzeige des Farbschrankes als auch für die Anzeige der Shopping List notwendig, da beide Seiten aus den eingescannten Items bestehen.



34 | **Farbschrank**



35 | **Shopping List**

Um zunächst die eigentliche Funktionalität zu erzeugen und zu testen, wurde vorerst eine vorgefertigte, mit fiktiven Daten gefüllte, JSON Datei erzeugt, die in der App eingebunden und angezeigt werden sollte. Die JSON-Datei enthält alle nötigen Inhalte, die für die App erforderlich sind:

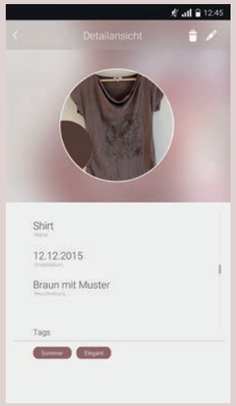
```
{
    „identification“:“1“,
    „bezeichnung“:“Pullover“,
    „beschreibung“:“Rentiermuster“,
    „geschaeft“:“S. Oliver“,
    „preis“:“19,99“,
    „imgURL“:“fotos/foto1.jpg“,
    „imschrank“:“1“,
    „farbe“:“rot“,
    „rot“:“80“,
    „gruen“:“56“,
    „blau“:“44“
},
```

Mit Hilfe von jQuery, welches ebenfalls zu Beginn in der index.html Datei eingebunden wurde, konnte in der script.js Datei problemlos das JSON Format ausgelesen werden. Durch den Einsatz einer for-Schleife kann das JSON-Array komplett durchlaufen werden. Auf diese Art und Weise kann für jedes Element ein entsprechender HTML-Content erzeugt werden. Hierzu wird auf die Funktionen .append und .html von jQuery zurückgegriffen. Mit diesen Funktionen ist es möglich HTML-Seiten und HTML-Content zu erzeugen und diese in der index. html Datei in einem angegebenen Bereich (z.B. einen div Container) anzuzeigen. An Stelle des selbst vorgegebenen Inhalts wird an dieser Stelle auf den Inhalt der JSON Datei zurückgegriffen und dieser entsprechend eingebunden und angezeigt.

```
for (var i = 0; i < itemArray.length; i++)
    {
    $('div.item-schrank').append(
    ,<a href="#" class="link-detail" data-id="'+itemArray[i].identification+'">'+
        ,<div class="item">'+
            ,<div class="item-image" style="background-image: url(,+itemArray[i].image+');">'+
                ,<div class="item-color">'+
                ...
                ,</div>'+
            ,</div>'+
            ,<p class="item-label">'+itemArray[i].bezeichnung+'</p>'+
        ,</div>'+
    ,</a>'
    );
```

Mit dieser Methode war es möglich den kompletten Inhalt des Farbschankes und den der Shopping List aus der JSON-Datei zu erzeugen und entsprechend dem geplanten Layout anzuzeigen.

2.3 Detailansicht



36 | **Detailansicht**

Im Anschluss an die erfolgreiche Realisierung der Farbschrank und Shopping List Ansicht mit dem zugehörigen Inhalt, galt es bei Tap auf ein Item die zugehörige Detailansicht aufzurufen. Bei der Umsetzung dieser Funktion bestand die Herausforderung darin, die Ansicht dynamisch zu erzeugen. Je nach dem welches Item der Benutzer auswählt, sollte sich der Inhalt der Detailseite passend zum zuvor ausgewählten Item erstellen.

Bei der Umsetzung half die jQuery .live() Funktion, die ein Event (in diesem Falle pagecreate) an ein HTML-DOM-Element bindet, welches erst nachträglich zur Seite hinzugefügt wird. Das Event in der Funktion wird ausgelöst, sobald der Nutzer auf ein Item (a.link-detail) in der Shopping List bzw. in dem Farbschrank klickt. In diesem Falle wird die Detailansichtsseite geladen und die ID des angeklickten Elements in der Variable „selectedItemForDetail" gespeichert. Anhand der Variablen kann das passende Objekt aus dem Array geladen und die Detailansicht erzeugt werden.

```
$('#schrank').live('pagecreate', function(e){
    $('a.link-detail').click(function(e) {
        $.mobile.changePage('#detail', { transition:'slide'});
        selectedItemForDetail = ($(this).attr('data-id'))-1;
    }
```

Es stellte sich jedoch heraus, dass die .live() Funktion veraltet ist und Probleme beim Seitenaufruf erzeugt. Aus diesem Grunde wurde nach einer anderen Möglichkeit gesucht, die Detailansicht zu erzeugen. Die letztendliche Lösung bestand darin, die Detailseiten mit den einzelnen Einträgen aus der Shopping List / des Farbschrankes zu erstellen. So sollte auch gewährleistet werden, dass die Verlinkungen passend gesetzt werden.

```
$('div.item-schrank').find('a.link-detail').click(function(e) {
    $.mobile.changePage('#detail', { transition: 'slide'});
```


2.4 Vergleichsfunktion

Die Grundfunktionalitäten des Farbschrankes und der Shopping List sind durch die oben aufgeführten Funktionen umgesetzt worden. Der nächste Schritt widmete sich der eigentlichen Hauptfunktion der App, dem Vergleichen zweier oder mehrerer Farbwerte. Um zunächst die einfache Funktion – das Vergleichen von Farben– zu realisieren und zu testen, wurde die entsprechende Rechenfunktion am Anfang erst extern angelegt. So konnte der Fokus auf das Wesentliche gelenkt werden und auftretende Fehler konnten besser identifiziert werden.

In einer zu Beginn leeren JavaScript Datei wurde zunächst die eigentliche Rechenfunktion erzeugt, welche den Prozentwert zwischen zwei RGB-Werten ermittelt. Hierzu wurden vorerst zwei RGB-Werte als globale Variable mit Beispielwerten definiert und entsprechend in der Rechenfunktion aufgerufen.

```
compareValuesInProcent =
((1 - (Math.abs(r - rV) / 255)) + (1 - (Math.abs(g - gV) / 255)) +
 (1 - (Math.abs(b - bV) / 255))) / 3 * 100;
```

Die oben aufgeführte Vergleichsfunktion ermittelt als erstes jeweils den Betrag der Differnz für R, G und B.

```
IR – R I= I r-Differenz  I
IG – G I=I g-DifferenzI
IB –B I= I b-Differenz I
```

Diese Differenz wird anschließend durch 255 geteilt (Math.abs(r - rV) / 255)), sodass eine prozentuale Abweichung resultiert. Die Übereinstimmung der Werte wird dementsprechend durch 1 minus den Wert errechnet:

```
1 – ( I r-Differenz  I / 255) = R Prozent
1- (I g-DifferenzI / 255) = G Prozent
1 – (I b-Differenz I / 255)  = B Prozent
```

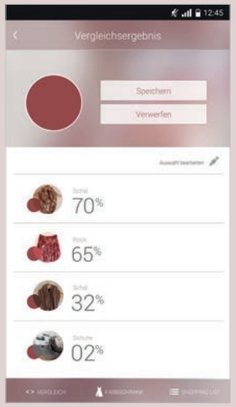
Im Anschluss werden die einzelnen Werte miteinander addiert und durch drei geteilt. So wird eine Gesamtübereinstimmung errechnet. Diese wird mit 100 multipliziert, sodass das Endergebnis aus einem Prozentwert besteht.

```
(R Prozent + G Prozent + B Prozent)  /3 * 100  = Übereinstimmung in %
```

Diese Rechnung wurde in der oben aufgeführten Vergleichsfunktion in einer Formel und in der richtigen Syntax zusammengefügt.

Um das Ergebnis zu überprüfen, wurde dieses in einem einfachen <p> Tag mit Hilfe von .append in der WebApp angezeigt.

2.5 Vergleichen mit mehreren Werten aus der JSON



37 | **Vergleichsergebnis**

Nachdem die Errechnung der Vergleichsprozentwerte mit den global definierten Variablen umgesetzt wurde, sollte im nächsten Schritt die zu vergleichenden Werte aus der JSON Datei geladen werden. Hierzu wurde die bereits erstellte JSON Datei, welche die Werte Rot, Grün und Blau der vorgegebenen Items enthält, genommen und wie oben beschrieben eingebunden. Mit Hilfe der for-Schleife, welche die gesamte JSON Datei (data.length) durchläuft, werden die RGB Werte von jedem Item aus der JSON genommen und in Variablen geschrieben.

```
var rV = data[i].rot;
var gV = data[i].gruen;
var bV = data[i].blau;
```

Diese Variablen wurden nun in der Vergleichsrechenfunktion eingefügt, sodass in dieser nicht mehr die global definierten RGB-Variablen, sondern die RGB-Werte aus der JSON mit dem festen RGB-Wert verglichen werden. Das Ergebnis wurde in der Variable “compareValuesInProcent“ gespeichert. Zur Anzeige der gesamten Werte wurde diese Variable bei jedem Schleifendurchlauf in ein neu erstelltes Array (compareValuesInProcentArray) geschrieben und anschließend in einer Liste angezeigt.

```
var compareValuesInProcentArray = new Array();
compareValuesInProcentArray.push(compareValuesInProcent);

for (var i = 0; i < compareValuesInProcentArray.length; i++)
    {
        $('#names-list').append('<li>'+'compareValuesInProcentArray
        + '%</li>' );
    }
```

Mit Hilfe von parseInt(compareValuesInProcent); wurde der angezeigte Float-Wert in einen Integer-Wert konvertiert, sodass die Prozentwerte ohne eine Nachkommastelle angezeigt werden. In einem weiteren Schritt wurde die Anzeige dahingehend optimiert, dass die Prozentwerte absteigend (höchste Prozentzahl zuerst) sortiert wurden.


```
compareValuesInProcentArray.sort(SortByCompareValue);
compareValuesInProcentArray.reverse();
```

Um die Werte den zugehörigen Items zuzuordnen, wurde das oben erstellte Array, welches in einer Liste dargestellt wird, um den Namen des Items erweitert.

```
compareValuesInProcentArray.push(compareValuesInProcent + data[i].bezeichnung;);
```

Diese Umsetzung stellte sich jedoch als etwas problematisch heraus. Da die Angaben zusammen in ein Array geschrieben wurden, wird ein String erzeugt, der nur umständlich wieder getrennt werden kann. Durch die Erzeugung dieses Strings, welcher mehrere Werte beinhaltet, konnte außerdem die zuvor erzeugte Sortierung nicht mehr korrekt angezeigt werden.

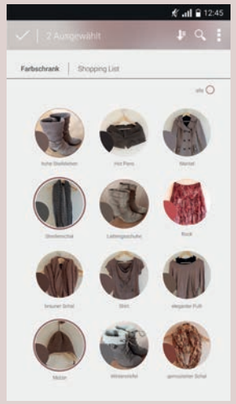
Als Lösung wurde an dieser Stelle die Erzeugung eines Objektes gesehen. Dieses sollte es möglich machen gleichzeitig verschiedene Variablen zu speichern. So wurde für jedes Item ein Objekt mit den Eigenschaften des Items angelegt und diese Objekte in einem Array gespeichert.

```
var compareItem = new Object;
compareItem.bezeichnung = data[i].bezeichnung;
compareItem.compareValue = parseInt(compareValuesInProcent);
compareItem.red = rV;
compareItem.green = gV;
compareItem.blue = bV;

compareValuesInProcentArray.push(compareItem);
```

Durch die Erzeugung eines Objekts konnte nun auf die einzelnen Variablen, die im Objekt enthalten sind zugegriffen werden. Dies sollte es möglich machen die geplante Sortierung nun wieder anhand der Vergleichswerte vornehmen zu können. Um dies zu realisieren wurde eine separate Funktion geschrieben (SortByCompareValue) welche bei der Sortierung aufgerufen wird und so die Werte wie gewünscht sortiert.

```
//Funktion zur Sortierung nach Vergleichswert im Objekt
function SortByCompareValue(a, b)
{
    return parseFloat(b.compareValue) - parseFloat(a.compareValue)
}

//Array wird nach den Prozentzahlen sortiert (Höchster Wert zuerst)
compareValuesInProcentArray.sort(SortByCompareValue);
```

2.6 Ausgewählte Items vergleichen



38 | **Items auswählen**
Farbschrank

Nachdem die RGB-Werte aus der JSON-Datei mit dem vordefinierten RGB-Wert verglichen und das Ergebnis im Browser in Prozent angezeigt werden konnte, bestand der nächste Schritt darin, nur bestimmte, durch den Nutzer ausgewählte Items (aus dem Farbschrank und/oder der Shopping List), mit dem RGB-Wert zu vergleichen. Hierzu wurde die vorhandene JSON-Datei zunächst um einen Eintrag „Selected“ pro Item erweitert, welcher den Wert „true“ oder „false“ mit übergeben bekommt. Der Grundgedanke bestand darin, eine If-Abfrage durchzuführen, welche die Vergleichsrechenfunktion nur bei den Items durchführt, bei denen Selected auf true steht ((if (selected == true))). Bei diesem Ansatz stellte sich jedoch schnell heraus, dass dies keine optimale Lösung darstellt. Die JSON-Datei müsste hier bei jeder neuen Auswahl neu geschrieben werden, was zu erhöhter Ladezeit führen würde.

Aus diesem Grunde wurde ein neuer Lösungsansatz erarbeitet, der sich mit der Kennzeichnung der Auswahl über IDs beschäftigt. Dabei werden die durch den Benutzer ausgewählten Items mit ihrer zugehörigen ID in ein Array geschrieben, welches anschließend mit den IDs aus der JSON-Datei abgeglichen wird. Wird eine Übereinstimmung der IDs festgestellt, wird für dieses Item die oben beschriebene Vergleichsberechnung durchgeführt und die Ergebniswerte entsprechend angezeigt. Zur Überprüfung dieses Ansatzes, wurde zunächst ein Array global angelegt, welches Testweise IDs der vom Nutzer ausgewählten Items beinhaltet.

Mit Hilfe zweier for-Schleifen können die IDs aus der JSON mit den IDs aus dem Array verglichen werden. Sobald eine Übereinstimmung vorliegt, wird die selbst erzeugte Variable „selected“ auf „true“ gesetzt.

```
for (var i = 0; i < data.length; i++)
{
    for(var j=0; j < selctedItemsIds.length; j++)
    {
     if(data[i].id == selctedItemsIds[j])
     {
        selected = true;
     }
}
```

Ist die Variable true wird die Vergleichsberechnung für dieses Item durchgeführt. Am Ende wird die Variable „selected” wieder auf „false” gesetzt.

Der gesamte Ablauf zum Vergleichen der Werte wurde im Anschluss in die Funktion (compareSelection();) überführt und in der $(document).ready Funktion, nachdem alle DOM-Elemente geladen wurden, aufgerufen. Auch das Anzeigen der Werte wurde in eine separate Funktion (showCompareValue();) geschrieben. Durch den Einsatz eines Objektes kann an dieser Stelle auf die einzelnen Variablen des Objekts zurückgegriffen und angezeigt werden.

```
function showCompareValue(){
    for (var i = 0; i < compareValuesInProcentArray.length; i++)
    {
        $('#names-list').append( '<li>'+'<div style="background-color:rgb
        ('+compareValuesInProcentArray[i].red+','+compareValuesInProcentArray[i].green+',
        '+compareValuesInProcentArray[i].blue+');"></div>'
        +compareValuesInProcentArray[i].bezeichnung +' ,
        + compareValuesInProcentArray[i].compareValue + ,%</li>' );
    }
}
```

Die ermittelten Werte sollten in der späteren App erst angezeigt werden, sobald der Benutzer seine Auswahl der Items bestätigt hat. Um auch diese Funktionalität zu realisieren und vorab zu testen, wurde zunächst ein einfacher Button angelegt, bei dessen Klick, die Funktion (showCompareValue()); zum Anzeigen der Werte, aufgerufen wird. Im Anschluss an die Anzeige wird das Array mit den Vergleichswerten wieder geleert.

Die oben beschriebenen Funktionalitäten wurden somit zunächst schrittweise und außerhalb der eigentlichen App programmiert und getestet. Mit Hilfe der Konsole und dem Befehl console.log wurden Fehler gesucht, analysiert und behoben.

Nachdem eine volle Funktionsfähigkeit der Vergleichsfunktion, dem Auswählen von Items und dem Anzeigen der Werte in seiner einfachsten Form, außerhalb des eigentlichen Projekts, erzielt wurde, galt es, diese nun in die eigentliche App zu überführen und die bisherigen Fakewerte (den vordefinierten RGB-Wert und das Array mit den ausgewählten Items) durch richtige Werte zu ersetzten.

39 | **Items auswählen**
Shopping-List

2.7 Optimierung der App durch den Einsatz von Objekten

Bevor mit der eigentlichen Überführung der Funktionen beginnen werden konnte, sollte zunächst eine Anpassung des bisherigen Codes in der eigentlichen App vorgenommen werden. Durch das Erarbeiten der oben genannten Funktionen wurde ein Verständnis für Objekten, dessen Erzeugung und Einsatz entwickelt. Das Arbeiten mit Objekten stellte sich als äußerst sinnvoll und effizient heraus. Aus diesem Grunde sollte der zuvor erzeugte Code dahingehend optimiert werden, dass nur einmal zu Beginn die JSON-Datei ausgelesen wird, welche anschließend ein Objekt erzeugt, das in ein Item-Array überführt wird. Diese Umwandlung wurde in eine Funktion „createItemArrayFromJson“ geschrieben, welche nun nur einmal zu Beginn aufgerufen werden muss und bei Veränderungen der JSON mit Funktion neu erzeugt werden kann.

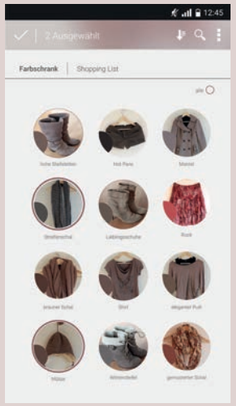
```
$.getJSON('json/items.json',
function(data)
{
    for (var i = 0; i < data.length; i++)
    {
        var item = new Object;
        item.identification = data[i].identification;
        item.bezeichnung = data[i].bezeichnung;
        item.beschreibung = data[i].beschreibung;
        item.store = data[i].geschaeft;
        item.price = data[i].preis;
        item.image = data[i].imgURL;
        item.inCloset = data[i].imschrank;
        item.farbe = data[i].farbe;
        item.red = data[i].rot;
        item.green = data[i].gruen;
        item.blue = data[i].blau;
        item.selectedItem = false;
        itemArray.push(item);
    }
});
```

Funktionen, die zuvor die JSON per for-Schleife durchlaufen haben, wurden dahin gehend angepasst, dass diese nun das Item Array durchlaufen und dessen Inhalt anzeigen.

Im Anschluss an die erfolgreiche Optimierung des Codes konnte die Aufmerksamkeit auf die Implementierung der zuvor extern erzeugten Funktionen gelenkt werden. Hierzu wurde zunächst die Vergleichsfunktion in die eigentliche App überführt und entsprechend angepasst.

Da die bisherigen Berechnungen teils noch auf bereits global definierten Werten basiert, sollten diese nun wie in der späteren App vorgesehen aus selbst erzeugten Werten bestehen.

2.8 Select Funktion



40| **Items auswählen**
Farbschrank

Zunächst wurde der Fokus auf das noch global definierte selctedItemsIds-Array gelegt, in dem bisher noch fest definierte Werte stehen und nicht die eigentlichen IDs der Items, die tatsächlich vom Nutzer ausgewählt wurden.

Der Benutzer soll in der App die Möglichkeit haben, zum einen alle Items auf einmal auszuwählen und zum anderen auch einzelne, jeweils aus dem Farbschrank und aus der Shopping List. Hierzu wurde zunächst die Page, wo der Benutzer Items auswählen kann dahingehend angepasst, dass sich hier nur der Inhalt der Seite austauscht, je nach dem, ob der Nutzer den Farbschrank oder die Shopping List ausgewählt hat. So bestehen beide Ansichten aus denselben Buttons und die Auswahl muss nicht neu geladen werden, wenn zwischen Shopping List und Farbschrank gewechselt wird. Hierzu wurde eine Funktion toggleCompareItem-View(); geschrieben in der, je nach dem auf welchen Button (Shopping List oder Farbschrank) der Nutzer klickt, entweder der Farbschrank angezeigt und die Shopping List mit der Klasse hidden versteckt wird, oder die Shopping List angezeigt und der Farbschrank mit der Klasse hidden versteckt wird.

Nun konnte sich der eigentlichen Funktion genähert werden. Um die Items, die durch den Nutzer ausgewählt werden, mit dem später eingescannten RGB Wert vergleichen zu können, muss zunächst überprüft werden, welche Items ausgewählt wurden. Hierzu wurde eine neue Variable „selectdItem“ dem zu Beginn erzeugten Objekt aus der JSON zugeordnet und bei der Initialisierung zunächst auf false gesetzt. An dieser Stelle wurde noch kein Item durch den Benutzer ausgewählt.

Sobald der Benutzer auf ein Item zum Vergleichen klickt, wird das Array mit den Objekten in der Funktion selectItem() durchlaufen. Hier wird wieder für jedes Objekt geprüft, ob die ID des jeweiligen Objekts mit der ID des geklickten Items übereinstimmt. Ist dies der Fall, wird die Variable selectedItem auf true gesetzt bzw. umgekehrt. Wäre die Variable bei dem Klick auf ein Item bereits auf true, würde es an dieser Stelle dementsprechend auf false gesetzt werden. Gleiches geschieht auch mit dem Style des Buttons. Ist der Button bereits ausgewählt, wird der Style dementsprechend bei einem Klick abgeschaltet.

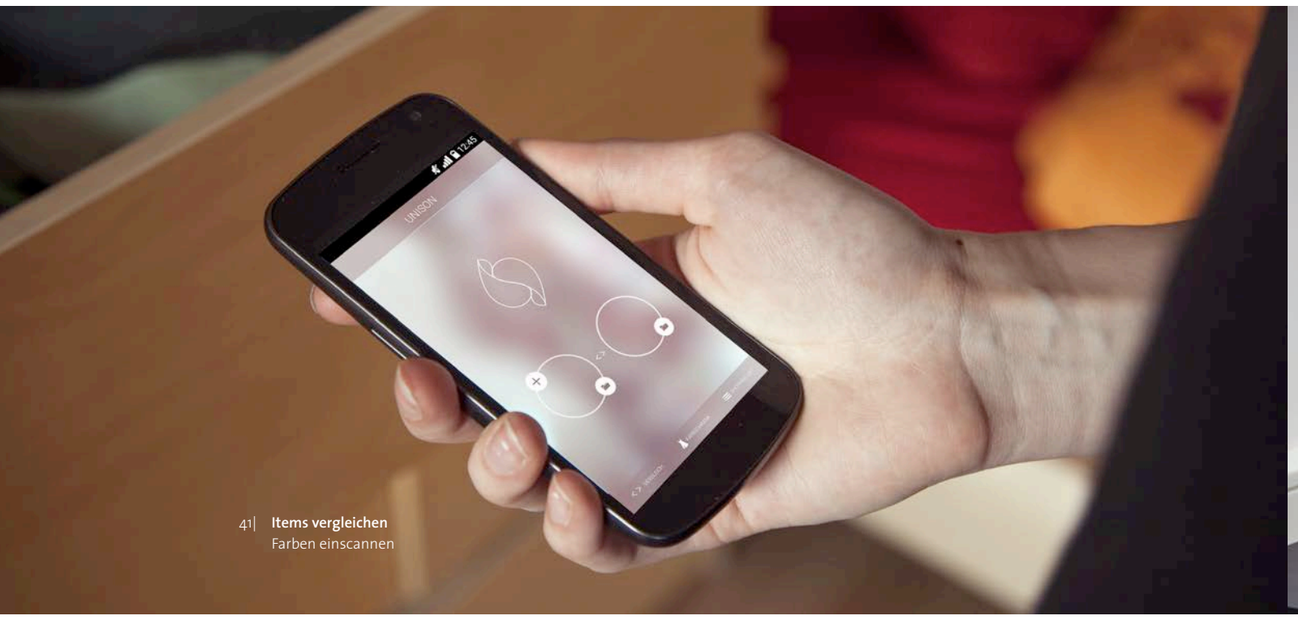
```
function selectItem(){
    $('.item-selected').click(function() {
        for (var i = 0; i < itemArray.length; i++)
        {
            if($(this).attr('data-id') == itemArray[i].identification)
            {
                itemArray[i].selectedItem = !itemArray[i].selectedItem;
                $(this).children('.item-image').toggleClass('selectedItemActive');
                $(this).children('.item-list').children('.item-image-list').toggleClass
                ('selectedItemActive');
            }
        }
    }
}
```

Sollte der Benutzer alle Items auswählen wollen, werden die Variablen aller Objekte auf true und der Style auf aktiv gesetzt, sobald die „alle“-Checkbox ausgewählt wurde. Wird die Checkbox deaktiviert, werden die Variablen entsprechend wieder auf false gesetzt und der Stil der Item-Auswahl wieder gelöscht. Diese Funktion wurde in der Funktion selectAllItems() definiert.

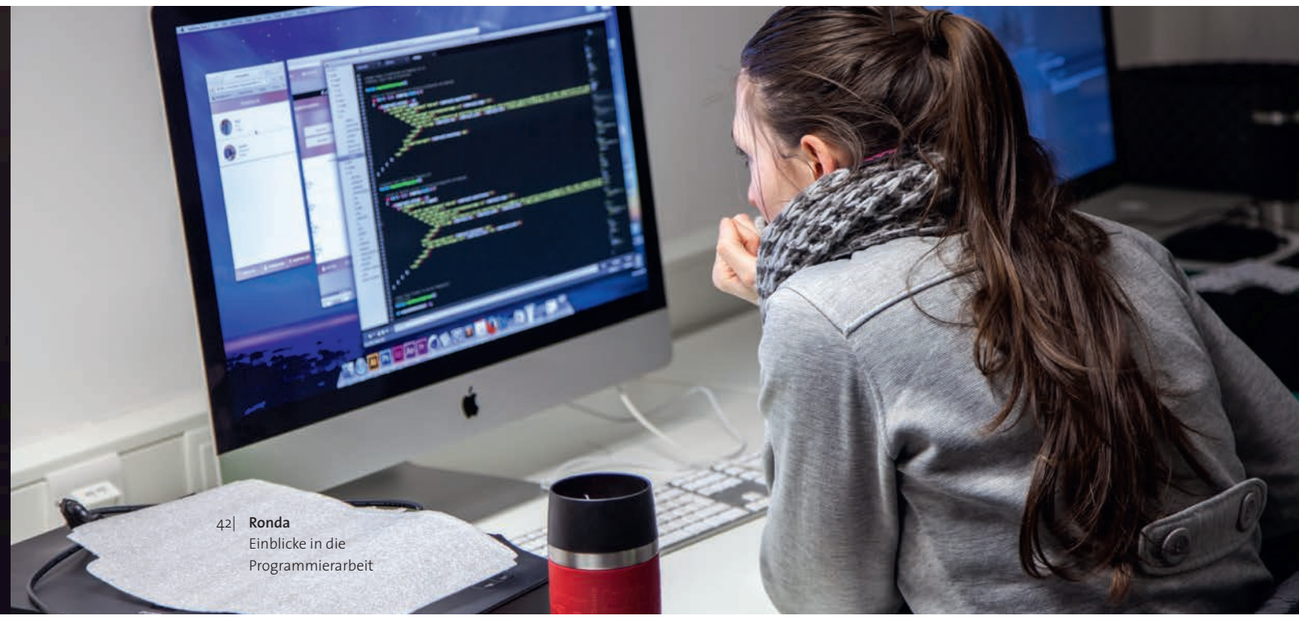
Im Anschluss wurde die bereits definierte Vergleichsfunktion so geändert, dass diese das ItemArray durchläuft und nur Objekte mit selectedItem == true vergleicht.

Nachdem ein problemloses Auswählen der Items realisiert wurde, sollte in einem weiteren Schritt die Anzeige der Ergebnisse korrekt eingebunden werden. Hierbei bestand der Anspruch, dass der Nutzer, wenn noch kein Item ausgewählt ist, lediglich einen Zurück-Button angezeigt bekommt. Sobald er jedoch ein Item auswählt, sollte der Zurück-Button durch einen Haken ersetzt werden, mit dem die Auswahl bestätigt wird und die Ergebnisse angezeigt werden. Um dies wie gewünscht umzusetzen, wurde zunächst eine Counter Variable (selectionCounter) eingeführt, die sobald ein Item ausgewählt wird um eins erhöht wird.

Ist diese Variable Größer als 0 wird der Back-Button durch den Check-Pfeil ersetzt. Ab diesem Zeitpunkt wird dem Nutzer ebenfalls angezeigt, wie viele Items dieser ausgewählt hat. Sobald die Variable kleiner als 1 ist und kein Item mehr ausgewählt ist, erscheint wieder der Back-Button und die Anzeige der ausgewählten Items verschwindet wieder.



41| **Items vergleichen**
Farben einscannen



42| **Ronda**
Einblicke in die
Programmierarbeit

2.9 Eingescannte RGB Werte zum Vergleich einbinden

Auf diese Art und Weise konnte die Auswahl der Items mit richtigen Werten erzeugt werden und die Ansicht so optimiert werden, dass der Nutzer zusätzlich über seine Auswahl informiert wird.

Nachdem diese gesamten Funktionalitäten erfolgreich realisiert wurden, fehlte jedoch noch ein wesentlicher Teil: das Vergleichen mit einem tatsächlich eingescannten Farbwert. An dieser Stelle wurde derzeit noch auf die global definierte RGB Variablen zurückgegriffen und die Items mit dessen Werten verglichen.

Das in Interaction Design realisierte Wearable Device verfügt über ein Bluetooth Modul, welches an dieser Stelle zum Einsatz kommen soll. Um eine Verbindung zwischen dem Smartphone und dem Arduino des Wearable Devices herzustellen, wurde auf das Bluetooth Plug-In von Cordova zurückgegriffen. Mit Hilfe von JavaScript konnte der String, den der Arduino übermittelt, ausgelesen werden. Der String besteht aus 4 Werten, die durch den Farbsensor erzeugt werden. Diese Werte werden durch den Arduino ausgelesen und, mit einem senkechten Strich getrennt, ans Handy gesendet.

Der Trennstrich wird dazu genutzt die 4 Werte voneinander aus dem String zu splitten und in ein dynamisches Array zu schreiben. Diese werden anschließend in einen Integer Wert gewandelt und in eine Variable geschrieben. Die Variablen enthalten nun die vom Sensor gemessenen Werte Rot, Grün, Blau und einen Helligkeitswert. Da der Sensor die RGB Werte nicht wie angenommen in Zahlen von 0-255 übermittelt, sondern in Zahlen von 0-1023, müssen diese entsprechend umgerechnet werden. Da der Sensors insgesamt 1024 verschiede Werte für RGB wiedergibt, das WEB-RGB System jedoch lediglich nur 255 Werte, müssen die Integer Werte entsprechend durch 4 geteilt werden.

```
var cutted = message.split('|');
var r = parseInt(cutted[0])/4;
var g = parseInt(cutted[1])/4;
var b = parseInt(cutted[2])/4;
var c = parseInt(cutted[3])/4;
```

Die drei Werte für R, G und B (c wird an dieser Stelle nicht mehr berücksichtigt, da dieser Helligkeitswert lediglich als Kontrollwert gilt) werden nun an zwei Funktionen übergeben.

```
colorCircle(parseInt(r),parseInt(g),parseInt(b));
sendColorValues(parseInt(r),parseInt(g),parseInt(b));
```

Mit Hilfe der Funktion colorCircle() wird der RGB Wert als Hintergrundfarbe für spezielle Divs mit der Klasse „farbwert“ in der App dem Nutzer angezeigt.

```
function colorCircle(newR, newG, newB) {
    $('.farbwert').css('background-color', 'rgb("+newR+","+newG+","+newB+")');
}
```

Die Funktion sendColorValues wird dazu genutzt, die eingescannten Werte aus der index.js zu lesen und als globale Variablen in die script.js zu laden, damit sie für die Vergleichsfunktion genutzt werden können.

```
function sendColorValues(newR, newG, newB) {
    r = newR;
    g = newG;
    b = newB;
}
```

Somit werden nun die angezeigten Vergleichswerte, mit den RGB Werten erzeugt, die vom Sensor ermitteltet werden.

2.10 Datenspeicherung

Bisher war es nur möglich die Daten aus einer externen JSON Datei anzuzeigen, da es Nutzern aber auch möglich sein sollte neue Items zu speichern, musste die Datenverwaltung überdacht werden.

Die Daten aus einer JSON Datei auszulesen war zu Beginn sehr einfach und komfortabel. Jedoch ergab die Speicherung der JSON Datei per Eingabeformular Schwierigkeiten. Die JSON Datei konnten nicht einfach per HTML/JavaScript gespeichert werden, da der Browser über keine Rechte zum Datei verwalten auf dem Computer verfügt. Eine Verwendung von PHP war auch ungeeignet, da sonst von der nativen App auf einen externen Server zugegriffen werden müsste und so zum Laden der Daten eine ständige Internetverbindung hergestellt werden müsste. Da die App vor allem offline funktionieren sollte, wurde diese Option ausgeschlossen.

Eine weitere Möglichkeit um JSON-Dateien zu speichern wäre die Verwendung des Cordova File Plugins gewesen. Da sich die Handhabung mit diesem Plug-In jedoch als schwierig herausstellte, wurden die Daten schließlich im Local Storage, dem internen Speicher des Browsers, gespeichert.

Der Local Storage eines Browsers ist zwar begrenzt, auf dem Mobil Device beträgt er meistens 5 MB, jedoch war diese Größe für die geringe Datenmenge vollkommen ausreichend. Der Vorteil von dieser Variante der Datenspeicherung ist, dass sie sowohl auf dem Smartphone, als auch im Browser funktioniert und so ein leichteres Debugging ermöglichte.

Beim Starten der App werden die Daten aus dem local.Storage geladen und, wie schon in der Vergleichsfunktion beschrieben, in einzelne Objekte geschrieben. Diese Objekte werden wieder in das itemArray geladen.

```
oldItemArray = JSON.parse( localStorage.getItem('daten'))
```

Das gesamte itemArray wird immer neu in den local.Storage geschrieben, wenn ein neues Kleidungsstück über das Formular hinzugefügt wurde.

2.11 Neues Kleidungsstück speichern

Damit der Nutzer die App mit seinem eigenen Kleiderschrank füllen kann, können neue Items mit Bild, Farbe und weitere Informationen über ein Formular gespeichert werden.

Immer wenn auf den Speichern-Button im Formular geklickt wird, wird ein neues Objekt angelegt. In dieses werden die Werte der einzelnen Formularfelder gespeichert und dem Objekt eine Identifikationsnummer gegeben. Je nachdem welcher Radiobutton im Formular vom Nutzer ausgewählt wurde, wird das Objekt im Farbschrank oder in der Shopping List gespeichert. Wenn der Farbschrank ausgewählt wurde, wird die Variable „inCloset“ auf 1 gesetzt, bei der Auswahl der Shopping Liste erhält sie den Wert 0. Diese Zuordnung ist für die spätere Anzeige wichtig, damit unterschieden werden kann, wo die Objekte angezeigt werden müssen.

```
$('#formular').submit(function(event){
    event.preventDefault();
    var closet;
    if ($('#radio-choice-1').is(':checked')) closet = 0;
    else closet = 1;
    item = {
        identification: itemArray.length+1,
        bezeichnung: $(,input[name="bezeichnung"]').val(),
        beschreibung: $(,input[name="beschreibung"]').val(),
        price: $(,input[name="preis"]').val(),
        geschaeft: $(,input[name="geschaeft"]').val(),
        red: $(,input[name="rot"]').val(),
        green: $(,input[name="gruen"]').val(),
        blue: $(,input[name="blau"]').val(),
        image: $(,input[name="imgURL"]').val(),
        inCloset: closet,
        selectedItem: false,
        };
});
```

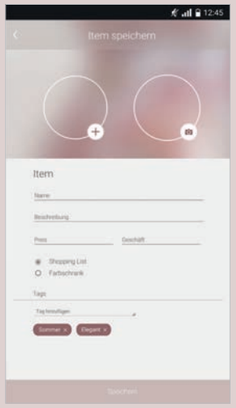
Schließlich wird das Objekt an das Array „itemArray“ von Objekten angehängt, sodass sich nun alle Objekte in einem Array befinden. Dieses Array wird anschließend in den localStorage gespeichert, damit die Daten auch noch nach dem Beenden der App vorhanden sind und neu geladen werden können.

```
localStorage.setItem('daten', JSON.stringify(itemArray));
```

Das neue Item wird an die Darstellungsansichten vom Kleiderschrank und der Shopping Liste weitergegeben, sodass ein neuer Eintrag für dieses Item erstellt und auf diesen Seiten angezeigt wird.

Zum Schluss werden alle Formularfelder wieder geleert und der Nutzer bekommt die Nachricht, dass sein Item gespeichert wurde. Anschließend wird er zu der Seite weitergeleitet, in die er sein Kleidungsstück gespeichert hat.

```
if (closet ==1 ) $.mobile.changePage('#schrank', { transition: 'slide'});
else $.mobile.changePage('#shopping-list', { transition: 'slide'});
```



43 | **Items erstellen**
mit passendem Foto

2.12 Cordova Kamera Plugin

Um die Kleidungsstücke schnell zu erkennen und eine schöne Übersicht zu erhalten, soll der Nutzer auch die Möglichkeit haben den Items ein Foto hinzufügen zu können.

Um auf die dazu erforderliche Kamerafunktion des Handys zugrückgreifen zu können, wurde auf das Cordova Camera Plug-In zurückgegriffen. Dieses lässt sich per Commando-Zeile in ein bestehendes Cordova Projekt integrieren. Mit Hilfe von JavaScript kann auf die Plug-In Funktionen zugegriffen und diese z.B. mit einem Button zum Auslesen der Kamera verbunden werden. Zu Beginn bereitete die Installation der Cordova Plug-Ins ein Problem, da diese im falschen Ordner installiert wurden. Dies führte dazu, dass in der Console vermehrt Fehler angezeigt wurden, die das Fehlen des Plug-Ins anzeigten. Um dieses Problem zu lösen, wurde ein neues Cordova Projekt über die Eingabekonsole angelegt, die Plug-Ins neu hinzugefügt und anschließend das komplette Cordova Projekt neu geladen. Im Anschluss befanden sich die Plug-Ins im richtigen Ordner und konnten verwendet werden.

Bei der Verwendung des Kamera Plug-Ins wurde auf einen bereits vorhandenen Code aus dem Internet zurückgegriffen, welcher die nötigen Funktionen zum Ansprechen der Kamera beinhaltete. Sobald der Benutzer auf den Button zum Hinzufügen eines Fotos klickt, wird durch die Funktion capturePhoto() die Handykamera aufgerufen und das Bild anschließend im Fotoalbum des Handys mit selbstdefinierten Maßen und Qualitätsangaben gespeichert. Zusätzlichen wird das erstellte Foto als Hintergrundbild in einen Div-Container geladen, sodass der Nutzer ein direktes Feedback über sein Foto erhält. Der Pfad des Bildes wird versteckt im Formular hinterlegt, sodass dieser mit gespeichert wird, sobald das gesamte Formular gespeichert wird.

2.13 Ladereihenfolge

An mehreren Stellen während der App Programmierung ergab sich das Problem, dass Items nicht angezeigt wurden, obwohl sie als Objekte angelegt und im Array gespeichert wurden. Schließlich stellte sich heraus, dass die Ansichtsseiten schon erstellt wurden, bevor die Daten komplett geladen und ins itemArray geschrieben wurden. Gelöst werden konnte das Ganze mit einem kurzen Ladescreen der beim Starten der App ausgeführt wird. Erst nachdem die kurze Animation abgelaufen ist, werden die anderen Funktionen über initializePage(); geladen.

Auch musste beachtetet werden, dass sich das itemArray ändert, wenn ein neues Item hinzugefügt wurde. Damit sichergestellt werden konnte, dass auf der Vergleichsseite immer das aktuelle itemArray angezeigt wird und nicht das, welches zu Beginn der App geladen wurde, muss die Vergleichsseite immer aktuell generiert werden. Hierfür wurde auf den JQuery Code, der schon in der Detailansicht beschrieben wurde, zurückgegriffen.

RESÜMEE

Die gesamte Entwicklung der hier erstellten Web App mit HTML5-/CSS- und Javascript ist eine wichtige Erfahrung, die das gesamte Wissen dieses und der vergangenen Semester im Bereich Webtechnologien forderte. Die gelernten Themen wurden hierbei nicht nur angewandt, sondern auch verstanden, erweitert und miteinander kombiniert. Ein wesentlicher Erkenntnisgewinn wurde im Bereich Objekte gewonnen. Diese zu erstellen und zu nutzen half dabei, gewünschte Funktionen besser miteinander kombinieren und wie gewünscht umsetzten zu können. Auch das Verständnis für den Bereich Datenhaltung wurde durch diese Hausarbeit geschärft. Insgesamt wurde bei der Programmierung viel mit dem Einsatz der Konsole und dem Befehl console.log() gearbeitet, mit dessen Hilfe schrittweise Fehler gesucht und behoben werden konnten. Des Weiteren wurden viele Funktionen zunächst extern erarbeitet und nach und nach erweitert. Diese schrittweise Erarbeitung führte zu ständig neuen Erkenntnissen, die den weiteren Verlauf der Programmierung beeinflussten. So wurden viele Funktionen und Ansätze im Verlauf verworfen und mit neuen Erfahrungen korrigiert und erweitert.

Im Endstadium der App stellte sich vor allem die Fehlersuche als schwierig und zeitaufwändig heraus. Durch die Kombination mit einem Wearable Devices konnten die letztendlichen Funktionen nicht ohne dieses in dem vollen Funktionsumfang getestet werden. Eine ständige Bluetooth Verbindung zwischen den beiden Geräten (Smartphone und Arduino) war notwendig, um mit dessen Hilfe die gescannten Werte an das Handy zu übermitteln. Diese Erfahrung bestätigte, dass es hilfreich ist, zunächst Funktionen mit festdefinierten Variablen zu testen, bevor die eigentlichen Werte zum Einsatz kommen.

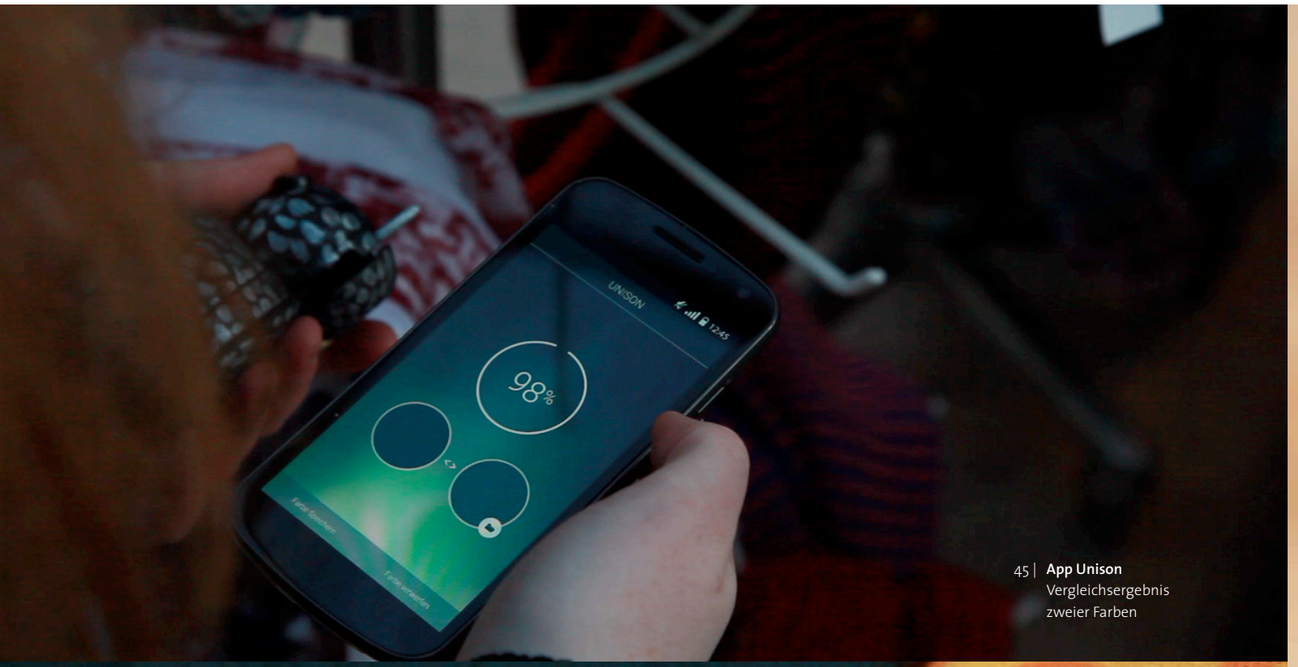
Insgesamt wurde die App mit Hilfe von zwei verschiedenen Smartphones entwickelt und getestet. Hierbei handelt es sich zum einen um das Samsung Nexus 4 mit der Android Version 4.2.2 und zum anderen um das Motorola Nexus 6 mit der Android Version 5. Bei der älteren Version von Android ergaben sich einige Anzeigefehler, die in der neuen Android Version nicht auftraten. Beispielsweise werden svg-Dateien nicht in der definierten Größe angezeigt und einige Textelemente mit einem anderen Stil versehen. Probleme, die nicht durch die Android Version erzeugt wurden, wurden sorgfältig untersucht und, so weit möglich, behoben.

In der hier erzeugten App konnten insgesamt die wichtigsten Funktionen wie gewünscht umgesetzt werden. Lediglich das Löschen von Items konnte aufgrund der Kürze der Zeit und der letztendlichen Komplexität des Codes nicht mehr realisiert werden.

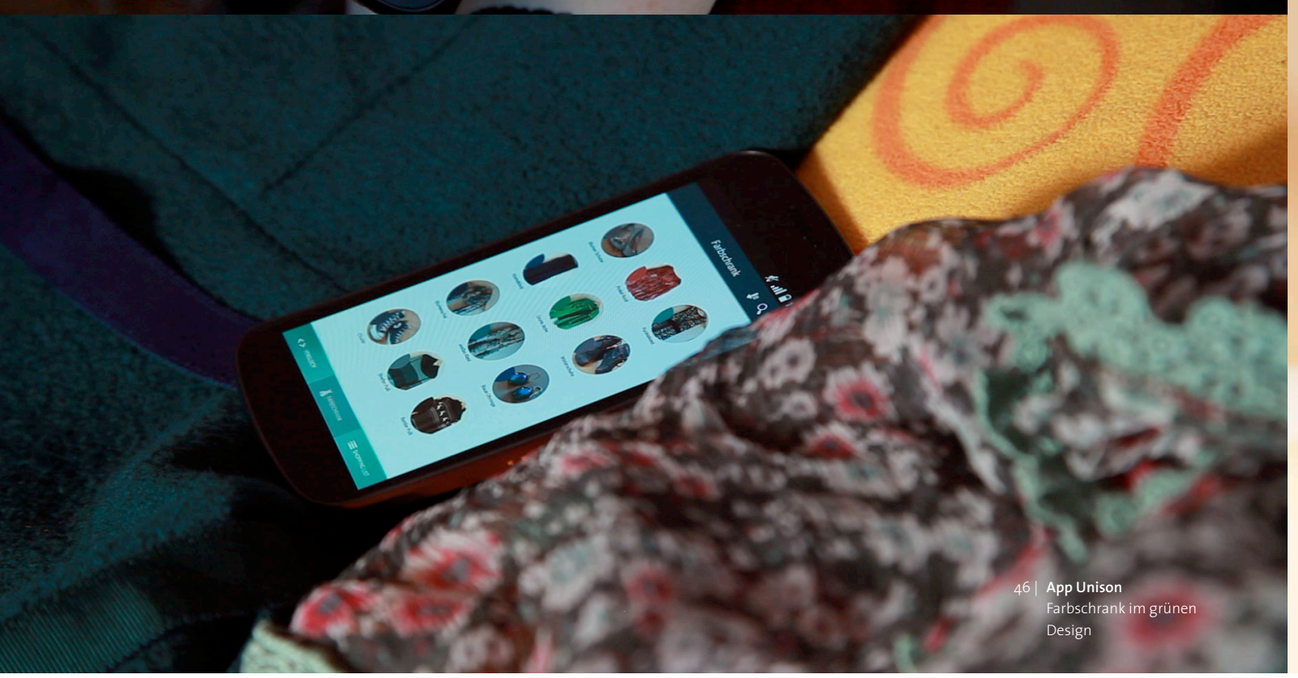
Als Ausblick für die App wäre eine automatische Farbanpassung der gesamten App, entsprechend der Farben aus dem Kleiderschrank, gewünscht. Diese angedachte Funktion wird auf Seite 60 bildlich verdeutlicht, dabei wird aus den eingescannten Farben, ein Hintergrundbild generativ erzeugt und das Farbschema der App dementsprechend angepasst. Außerdem würde eine Optimierung der Vergleichsfunktion, durch den Einsatz von HSB Werten, die Aussagekraft der Vergleichswerte sehr wahrscheinlich stärken und bessere Werte hervorringen. Bei den RGB Werten wird bisher der gesamte Farbwert verglichen und weniger der eigentliche Farbton.

Abschließend ist zu sagen, dass die Umsetzung der App einige Schwierigkeiten und Herausforderungen mit sich brachte, die aber gemeinsam im Team erarbeitet und größtenteils behoben werden konnten. Jedes Teammitglied brachte seine Erfahrungen mit, die im gemeinsamen Austausch und durch gemeinsame Erweiterung des Wissens zu der hier erstellten App beitrugen.

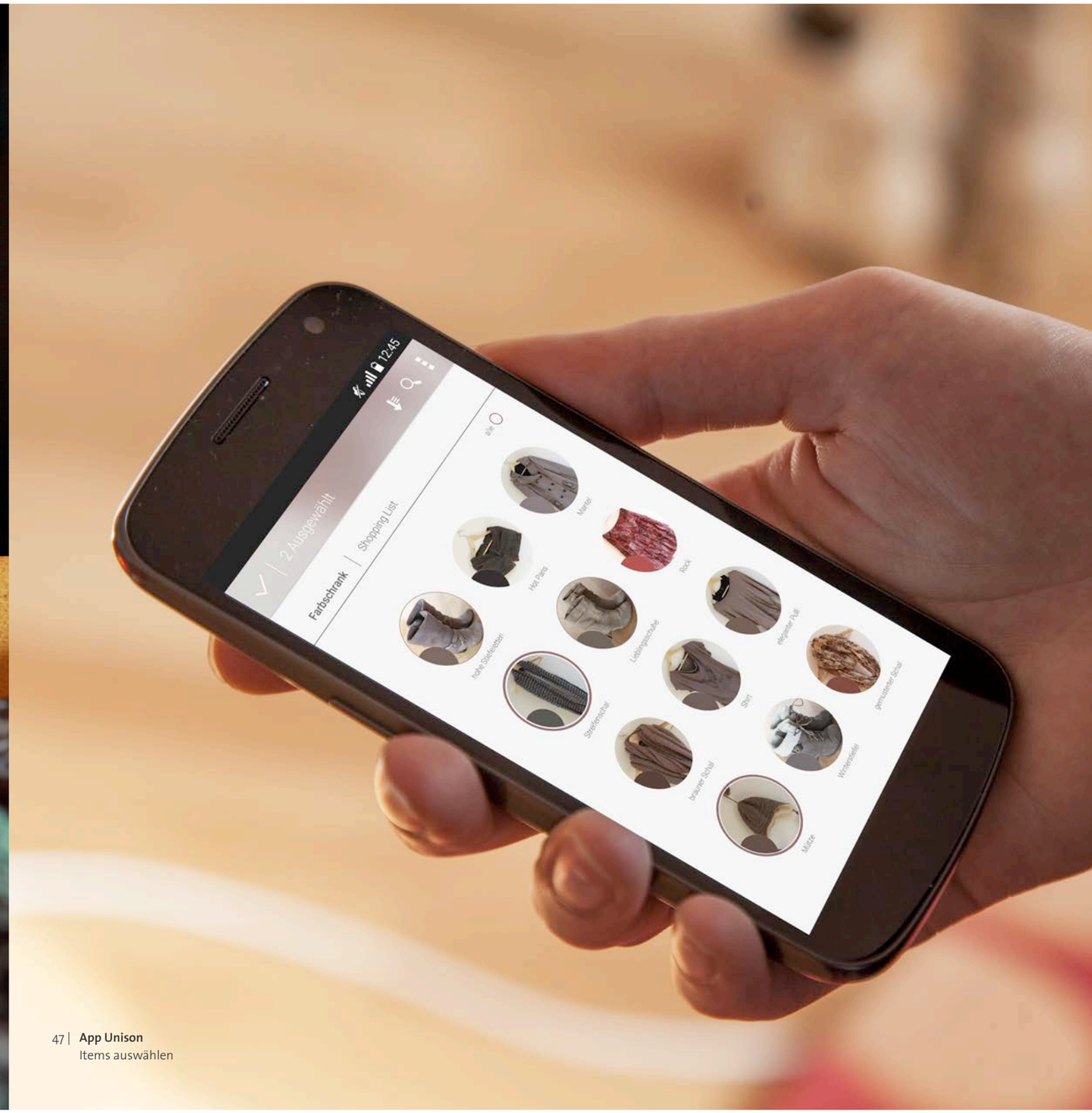
44 | **Mobil Device & Wearable**



45 | **App Unison**
Vergleichsergebnis
zweier Farben



46 | **App Unison**
Farbschrank im grünen
Design



47 | **App Unison**
Items auswählen

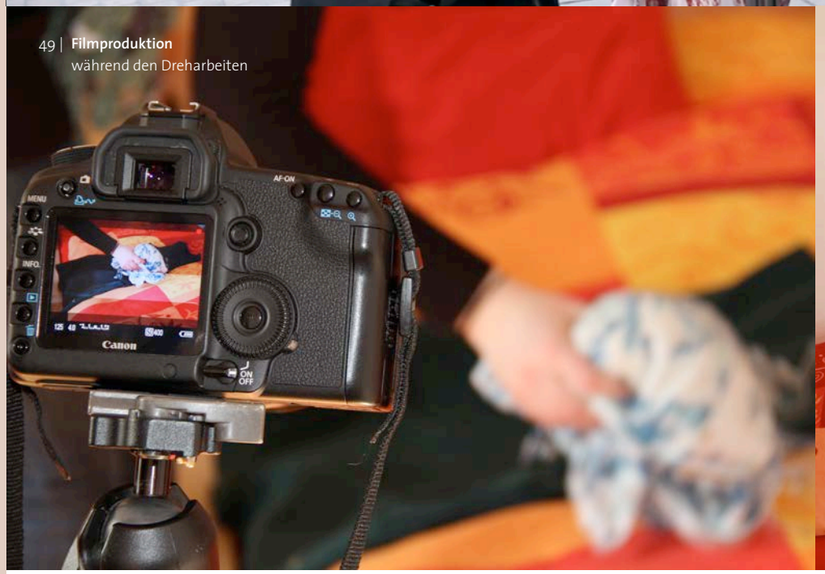
UNISON
FILM

Um die Benutzung des Wearable Devices zusammen mit der App so gut wie möglich zu visualisieren und auch digital präsentieren zu können, wurde ein Kurzfilm gedreht. Dieser stellt ein Problem vor, mit dem sich Unsion auseinandersetzt und wie dieses mit Hilfe der App und des Wearables gelöst werden kann.

Story
Der Film zeigt, wie mit Hilfe von Unison ein farblich passender Schal zum Kleid gefunden wird. Zunächst wird dafür der Charakter vorgestellt, der in die Zielgruppe von Unison passt, eine Frau Mitte 20. Danach wird das Problem vorgestellt, nämlich dass diese keinen passenden Schal zu ihrem Kleid findet. Auf die Lösung dieses Problems muss nicht lange gewartet werden, da die Hauptfigur schnell zum Handy greift und per App ein neues virtuelles Kleidungstück erstellt. Anschließend wird das Wearable vorgestellt, mit dem die Farbe des Kleidungsstücks eingelesen wird. Das Handy und das Wearable werden nach einer kurzen Überleitung beim Shoppen erneut benutzt um die Farbe eines Schals einzuscannen und zu vergleichen. Da der Vergleich eine sehr hohe Übereinstimmung ausgibt, kann sich die Protagonistin glücklich für den Kauf entscheiden.

Umsetzung
Zunächst wurde die Story mit Hilfe der Definition eines typischen Nutzungsszenarios festgelegt. Dabei wurde darauf geachtet, dass die Protagonistin, das Problem und die Lösungsfindung durch Unsion nacheinander verständlich präsentiert werden. Dazu wurde anschließend ein Storyboard erstellt, welches diskutiert wurde und in einer zweiten Version dem Drehort angepasst wurde. Der Dreh fand an zwei Orten statt, in einer hellen Wohn-/Schalfzimmer Umgebung und in der Innenstadt. Die verschiedenen Handlungsabläufe wurden dort in verschiedenen Einstellungen gedreht und anschließend zur Musik passend geschnitten. Es wurde dafür eine sehr fröhliche Musik gewählt, die zum Thema Life Style passt und die Hauptaussage, wie schnell und einfach Unsion eine Lösung bietet, unterstützen soll. Die Animationen der App wurden zuvor mit Hilfe des Storyboards erstellt und auf dem Handy abgespielt. Dabei wurde die optimale Version der App visualisiert, dessen Farben sich an denen des Kleiderschranks anpassen.

48 | **Nutzungsszenario**
Screenshots aus dem Film



49 | **Filmproduktion**
während den Dreharbeiten

50 | **Filmproduktion**
Unter erschwerten Bedingungen.

Unison

Jasmin Bleeke – 626504
Dorena Diekamp – 627910
Ronda Ringfort – 628885