Smart Mirror

Baris,Tikir, Leon Dodrimong

1. März 2020

Inhaltsverzeichnis

1 Grundidee

Wie wir auf die Idee gekommen sind... Auf die eigentliche Idee ist Barsi Tikir eines morgens gekommen als er im Bad stand. Während er sich für den Tag fertig machte und auf seinem Handy noch seine passsende Bahnverbindung raus suchte, dachte er darüber nach, wie praktisch es wäre, wenn man seine Bahnverbindungen nicht mühsam im Handy über die BVG App suchte müsste, sondern diese im irgendwie direkt präsent wäre. Als er daraufhin in den Spiegel blickte kam im die Idee.. ein Spiegel der seine Bahnverbindung anzeigen könnte. Morgens In den Spigel gucken, beim Zähneputzen, Haare machen, usw.... Warum nicht die Zeit auch geich nutzen für ein kleines Update, was in der Welt gerade so passiert oder wann der nächste Bus zur Arbeit fährt. Nach kurzer Recherche fand er auch eine passende Bauanleitung für einen solchen Spiegel. Jedoch gab es neben dem Zusammenbau noch das Problem etwas sinnvolles auf dem Display anzuzeigen. kleinere Projekt mit Uhrzeit und Wetter gab es bereits, als Beispiele zum nachprogrammieren. Aber ein wirkliches System oder fertiges Endprodukt fand er nicht.
Ein paar Wochen später stieß er auf den Paulaward. Als wir uns trafen, erzählte Baris über die Idee und den Paul Award. Wir fantasierten ein bischen rum und überlegten uns, was denn alles möglich wäre. Zusammen haben wir uns dann rangesetzt und ein Konzept entwickelt, welches möglichst viele Infomationen aus verschiedenen Bereichen anzeigen kann. So entstand die Idee vom SSmartMirror".

1.1 Spiegel mit integriertem Display

Ein Spiegel mit integriertem Display ist wohl nichts wirklich neues. Die Idee ist einfach hinter einem Einwegspiegelglass ein Display zu montieren, sodass dieses durch das Glass durch scheint und man dies auf der anderen Seite des Spiegels sehen kann. Von der Anderen Seite wirkt das Galss spiegelnd, wodurch es ganz Normal als Spiegel genutzt werden kann.

1.2 Gesamtkonzept - eigentliche Idee

Unsere eigentliche Idee von uns hinter einen SSpiegel mit integriertem Display"(wird werden ihn im folgenden SSmartMirrornennen, so wie auch unser Projekt heißt), ist einen System für den SmartMirror zu entwickelt, welches zum einen Benutzerfreundlichkeit(mehr dazu siehe Kapitel 1.2.1) aufweist und zum Anderen mit möglichst vielen Systemen komatibel ist. Wir wollten nicht ein in sich geschlossenes System entwickeln, welches vielleicht gut funktionieren würde, sondern auch ein System, welches bereits vorhandene Dienste nutzen kann. Mehr dazu haben wir im Kapitel 1.2.2. Wir trafen uns

meist nach dem Studium und haben uns viele Gedanken darüber gemacht, was der SmartMirror können sollen und über dessen Umsetzung.
Wir stellte einige Grundfunktionen auf die aus unser Sicht wichtig für den SmartMirror wären und welche zusätzlich auch noch sehr praktikabel wären. In diesem Dokument konzentrieren wir uns eher auf die wichtigen Grundfuktionen (Mehr dazu siehe Kapitel 4 Funktionen).
Danach versuchten wir ein Konzept zu entwickeln, welches allen Anforderungen entspricht. Dabei war es uns sehr wichtig auf den Punkt Skalierbarkeit zu achten.

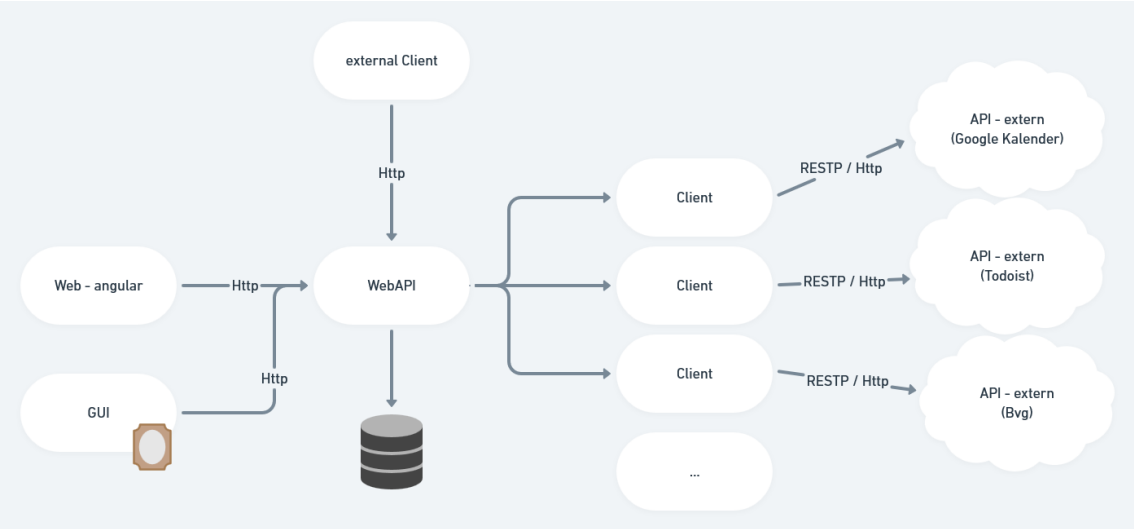

Abbildung 1: konzeptioneller Aufbau der Software-Komponenten

Nach meheren Skizzen und Umgestaltungen kamen wir dann auf diesen Aufbau unserer Software. Auf dem RasperbryPi laufen die folgenden Komponenten: Web - angular, GUI,WebApi, Datenbank und die Clients (mehr zu den einzelnen Komponenten finden sie im Kapitel 3.4).
Die Clients übernehmen die Kommunikation zu den externen Dienst-Anbietern über deren APIs. Die WebAPI organisiert die gesamten anfallenden Daten, wie zum Beispiel: Client-Daten, Benutzerdaten und GUI-Elemente. Auserderm organisiert sie die einzelnen Clients mit deren dazugehörige API und stellt ein eigene API bereit über die Daten abgefragt werden können. Diese API nutzt wiederrum das Frontend, also die GUI Komponente und die Web Komponente. Die GUI ist für die Ansicht auf dem Display hinter dem Glass zuständig. Sie fragt die Daten vom aktuellen Benutzer über die WebAPI ab und formt diese Daten zu einer Ansicht. Die Web Komponente stellt einen Web-Service zur Verfügung, sodass der Benutzer über ein Web-Interface über ein mobiles Endgerät Einstellungen vornehmen kann. Dort kann er Einstllungen bezüglich verschiedener Benutzerkonten und de-

ren Dienste machen. Diese API kann natürlich auch von externen Clients benutzt werden, um zum Beispiel mit eigene SmartHome Produkte mit dem SmartMirror zu interagieren oder den SmartMirror zu steuern.

Um den SmartMirror (RaspberryPi) ins eigene WLAN zu bringen haben wir uns folgende Methodik überlegt. Der RaspberryPi stellt über den integrierten Wifi-chip ein eigenes WLAN zur Verfügung. In dieses muss sich der Benutzer zur Konfiguration einwählen. über ein kleines Web-Interface soll er dann sein WLAN auswählen und das Passwort eingeben oder den SmartMirror über den eigenen WLAN Router freischalten.

Clients Die Clients, wobei jeder auf eine spezifische externe API ausgreichtet ist, übernehemen die Kommunikationion zu diesen. Sie kümmern sich um die Authorisierung, Datenabfrage und stellen der WebAPI bestimmte einheitliche Funktionen bereit. Jeder Client muss diese implementieren. Zusätzlich ist der Aufbau der Daten, die zwischen der WebAPI und den Clients übertragen werden, vereinheitlicht und fest definiert. Diese vereinheitlichten Funktionen und Datenmodelle sind in von uns definierten Interfaces beschrieben. Ein Client muss sich an die vorgebenen Funktionen und Datenmodelle halten. Dabei haben wir geguckt, welche Funktionen essentiell wichtig für die Kommunikation sind und wie wir die Daten optimal strukturieren können. (einige Funktionen um nicht zu sehr ins Detail zu gehen zb.: getData(Options):Data, welche Daten von der API und den Bedingungen abfragt; getOptions():Options, welche die Verfügbaren Optionen abfragt; setToken(string):bool, welche den Authorisierungstoken für den jeweiligen Client setzt). Wenn eine neue externe API bereitgestellt werden soll, muss lediglich ein Client programmiert werden, welcher die vorgeschriebenen Funktionen und Datenmodelle implementiert (Interfaces) und die jeweiligen Http-Requests für die API beinhaltet. So kann zu jeder API unter berücksichtigung der Interfaces ein passender Client programmiert werden.

WebAPI Die WebAPI übernimmt die

1.2.1 Benutzerfreundlichkeit

Die eigentliche Idee dabei ist den Spiegel nicht einfach nur spiegeln zu lassen oder die Uhrzeit anzeigen zu lassen, sondern ihn ßmartßu machen, sodass man ihn praktischen und effizient Nutzen kann. Zudem war uns auch wichtig, dass es **Benutzerfreundlich** ist, da nicht jeder das Know-How hat sich einen Spiegel für seine Eigenen Bedrüfnisse zusammen zu bauen bzw. zu programmieren. Er sollte einfach und verständlich für jeden sein. Deshalb haben wir auch die in das Konzept die Web Komponente eingefügt und die GUI leicht erweiterbar gestaltet.

1.2.2 Skalierbarkeit

Der SmartMirror ist nicht nur Benutzerfreundlich, sondern ist auch in vielen Richtungen skalierbar. Wie schon erwähnt, wollten wir die bereits bestehenden Dienste Nutzen. Zum Beispiel hat man seine Kalendereinträge bereits im Google-Kalender eingetragen und hat keine Lust als Kunde alle Daten auf den Kalender des SmartMirrors zu übertragen. Es wäre für den Benutzer leichter auf diese Daten zuzugreifen.
Viele Anbieter von Diensten stellen über das Http-Protokoll API Funktionen für Ihren Dienst bereit. Diese API's nutzt der SmartMirror, um dann die Daten vom dem jeweiligem Dienst abfragen zu können. Dieses Abfragen geschieht in der Client Komponente, auf welche wir näher im Kapitel 3.4.1 Clients eingehen werden. Dadurch ist das System erweiterbar und hilft sogar beim Vernetzen von anderen Diensten.
Neben der Erweiterbarkeit haben wir auch daran gedacht, eine eigene Schnittstelle zu definieren, sodass andere SmartHome-Produkte sich an den SmartMirror vernetzen können. Diese können dann wie die GUI und Web Komponente, Daten vom SmartMirror abfragen oder senden, sodass es zum Beispiel auch denkbar wäre mehrere SmartHome-Produkte so miteinander zu vernetzen, dass der SmartMirror die Basis bildet.
Im Sinne von Smart Home und Industrie 4.0

2 Aufbau

dem Aufbau der Software haben wir aufgrund der Vielfalt ein eigenes Kapitel gewidmet. (3)

2.1 Hardware - Spiegel

2.2 Materialien

Der Spiegel besteht aus einem Einwegspiegelglass, welches von einer Seite spiegelt und von der anderen Seite reflektiert. Außerdem haben wir einen Holzrahmen, welcher aus Fassung für das Spiegelglass dient, genutzt. Für die Technik haben wir ein Display für die Anzeigen , ein RasperryPi für die Steuerung und die nötigen Verbindungskabel, wie Spannungsversorgung und Videokabel (HDMI) zur Übertragung der Videosignals zum Display, eingesetzt.
Für die optinalen Erweiterungen würde man je nachdem welches Feature gewünscht ist, noch eine Picamera (für Facerecognition), RasperryPi Bewegungssensor (Bewegungserkennung)[1], Gesture Sensor (Gestiksteuerung)[2]

[1] *Raspberry Pi Infrarot Bewegungsmelder:* https://www.reichelt.de/raspberry-pi-infrarot-bewegungsmelder-hc-sr501-rpi-hc-sr501-p224216.html?&nbc=1

[2] *3D Gesture Tracking Shield for Raspberry Pi:* http://wiki.seeedstudio.com/3D-Gesture-Tracking-Shield-for-Raspberry-Pi-MGC3130/

oder ein kleines Mikrophone zur Sprachsteuerung [3]

Kosten Da der SmartMirror für Jederman sein soll, darf dieser auch nicht das Budget eines Einzelnen sprengen. Daher hatten wir auch im Hinterkopf, dass die Koponenten nicht zu teuern sein dürfen.
Der RasperryPi kostet ca. €35, Das Glass ist etwas teurer mit ca.€80. Der Rahmen ca. €10 - €20 je nachdem welches Design gewünscht ist. Mit Kabeln ca. €10 läge der Gesamtpreis für das Material bei ca. €140. Wenn man überlegt, dass ein Normaler Spiegel ca. €80-€100 kostet, recht preiswert ist.

3 Software

3.1 Prerequisites

- Node.js (Javascript runtime)
- Angular
- IDE (Visual Studio Code)
- Datenbank (MariaDb)

3.2 Frontend - SmartMirrorWeb

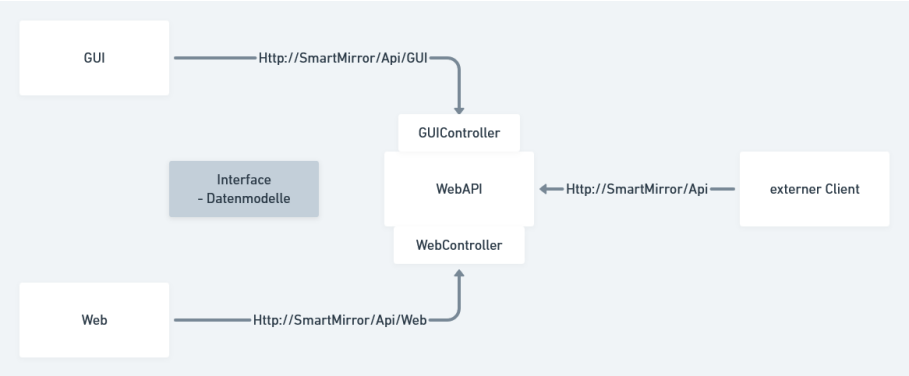

Abbildung 2: konzeptioneller Aufbau der Software-Komponenten

Wir haben unser Frontend mithilfe von Angular 9 entwickelt. Für uns war dies eine Neue Erfahrung, da wir noch nie zuvor damit gearbeitet haben.

[3] *Ansteckmikrofon über Klinke:* https://www.amazon.de/dp/B073GJQKL1/ref=psdc_1384055031_t1_B07WQFNVVQ

Das Frontend soll sich lediglich um die Anzeige der Daten kümmern und folgende Funktionen übernehmen:

- Daten vom Backend (SmartMirror.WebApi) anfragen
  - alle möglichen Dienste (Widgets)
  - vom User angemeldeten Dienste
- Daten senden
  - Dienst aktivieren/ deaktivieren
  - Benutzereingaben zur Erstellung eines neuen Benutzers
- User zur Anmeldung von weiteren externen Diensten, zur jeweiligen Website weiterleiten[4]

3.3 Backend - SmartMirror.WebApi

Das Backend besteht aus einer eigens Entwickelten Web API, welche zum einen die Anfragen und Daten vom Frontend (SmartMirror.Web) entgegennimmt und bearbeitet, und zum anderen den Datenaustausch mit der eigenen Datenbank und Kommunikation mit den externen API kommuniziert.
Das Backend besteht zum einen Client-Server. Zum einen stellt er als Server eine eigene API dar, welche vom Frontend genutzt wird. Zum anderen fungiert dieser auch als Client und nutzt die externen API Schnittstellen von Drittanbietern. Besitzt dieser eine extra Komponente, welche auf die interne Datenbank zugreift

Server Funktion

Kommunikation zur Datenbank

3.4 Komponenten

3.4.1 Clients

Da der Smart Mirror möglichst viele externe Features verbinden soll, muss die Schnittstelle zu diesen gut strukturiert werden. Daher haben wir uns dazu entschieden, dass jede externe Kommunikation ihre eigene Komponente bekommet, welche bestimmte Standards (Interfaces) bedient Die Clients, wobei jeder auf eine spezifische externe API ausgreichtet ist, übernehemen

[4] zum Beispiel: wenn der User den neuen Dienst Google Kalender für sich registrieren möchte, muss er sich auf der Website von Google Kalender anmelden um sich zu zertifizieren. Diese sendet dann die zur Authentifizierung notwendigen Credentials, welche vom Backend gespeichert werden müssen

die Kommunikationion zu diesen. Sie kümmern sich um die Authorisierung, Datenabfrage und stellen der WebAPI bestimmte einheitliche Funktionen bereit. Jeder Client muss diese implementieren. Zusätzlich ist der Aufbau der Daten, die zwischen der WebAPI und den Clients übertragen werden, vereinheitlicht und fest definiert. Diese vereinheitlichten Funktionen und Datenmodelle sind in von uns definierten Interfaces beschrieben. Ein Client muss sich an die vorgebenen Funktionen und Datenmodelle halten. Dabei haben wir geguckt, welche Funktionen essentiell wichtig für die Kommunikation sind und wie wir die Daten optimal strukturieren können. (einige Funktionen um nicht zu sehr ins Detail zu gehen zb.: getData(Options):Data, welche Daten von der API und den Bedingungen abfragt; getOptions():Options, welche die Verfügbaren Optionen abfragt; setToken(string):bool, welche den Authorisierungstoken für den jeweiligen Client setzt). Wenn eine neue externe API bereitgestellt werden soll, muss lediglich ein Client programmiert werden, welcher die vorgeschriebenen Funktionen und Datenmodelle implementiert (Interfaces) und die jeweiligen Http-Requests für die API beinhaltet. So kann zu jeder API unter berücksichtigung der Standards ein passender Client programmiert werden. Anschließend muss dieser nur noch der WebAPI bekannt gemacht werden, indem der Client in eine interne Liste von allen möglichen externen Diensten eingetragen wird.

3.4.2 WebAPI

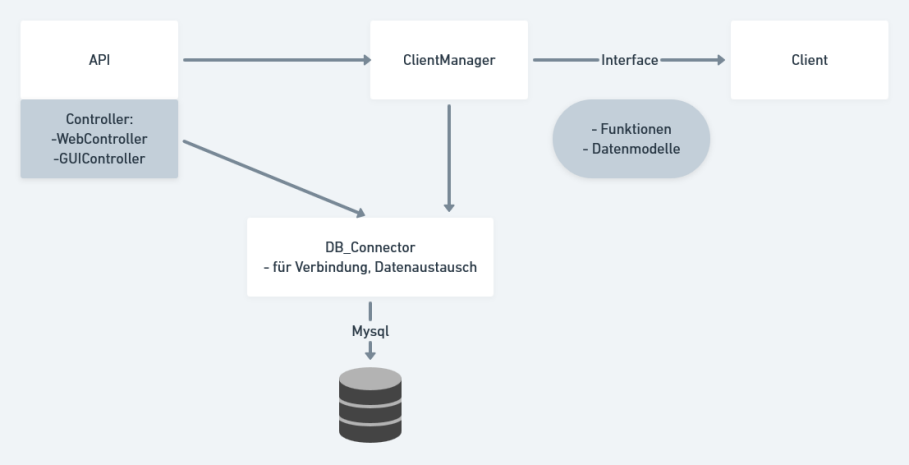

Abbildung 3: konzeptioneller Aufbau der Software-Komponenten

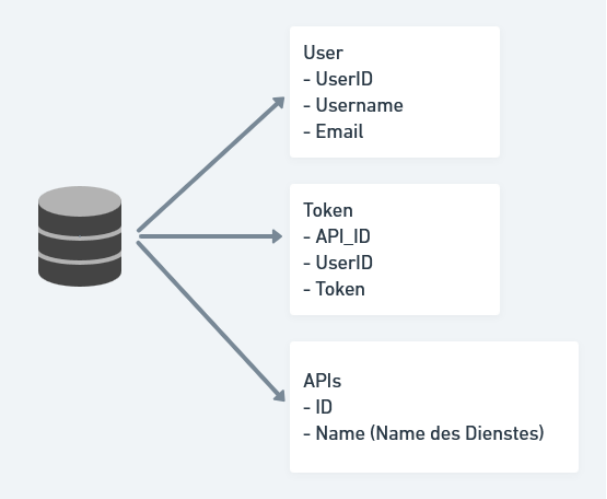

Abbildung 4: konzeptioneller Aufbau der Software-Komponenten

3.4.3 Database

4 Funktionen

4.1 Grundfunktionen

- Daten von externen API's abfragen
- Daten von externen API's anzeigen (Widgets)
- Einstellungen über ein Web Frontend vornhemen
- Schnittstelle für andere Erweiterungen (externe Clients)
- ...und natürlich auch die Grundfunktion spiegeln nicht vergessen

4.2 Extras

- Face Recognition
- Voice Control
- Bewegungssensor

5 Tests

Wir haben bis zum letzten Entwicklungsstand vor der Abgabe jede Komponente Einzeln getestet. Die Funktionen die wir bis zu dem Zeitpunkt implementiert haben, funktionierten. Das Zusammenspiel einzelner Komponenten

wurde ebenfalls getestet durch extra angefertigte Test. So haben wir zu Beispiel ein Test Projekt angelegt in dem wir den im Frontend benutzten API Client für unsere WebAPI testen konnten, indem wir die Datenbank mit Beispiel Daten belegten und dann einige Testanfragen laufen ließen.
Wir verbrachten viel Zeit mit der Konzeptionierung, dem Entwurf und vorallem mit der Informationsbeschaffung, da wir vor diesem Projekt mit vielen Techniken noch keine Erfahrungen gemacht haben und uns einiges erst noch beibringen mussten.
Leider konnte der komplette SmartMirror nicht komplett getestet werden, da uns einfach gesagt die Zeit fehlte. Wir werden das Projekt jedoch in Zukunft fortsetzen.
Das Projekt befinden sich auf **Github** in folgendem Repository:

https://github.com/brstkr/SmartMirror.git

6 Über uns

Wir sind zwei Studenten, Baris Tikir und Leon Dodrimong, von der HTW - Hochschule für Technik und Wirtschaft Berlin aus dem Fachbereich der Ingenieurwissenschaften und Technik.

7 Anhang

7.1 Verweise