Projektdokumentation Gif - Fotobox

MOSY / ITS SS 2017 – Torsten Edeler / Andreas Plaß – HAW Hamburg – 20.07.2017

Raoul Bickmann (2217470)
Lukas Heiduk (2248591)
Hennes Römmer (2217606)
Mia Wölm (2221086)

Idee

Das Konzept der Fotobox lehnt sich an den Fotoboxen an, die gerne auf Feierlichkeiten wie zum Beispiel Hochzeiten genutzt werden. Mit der Fotobox kann der User über sein Handy ein bewegtes GIF aufnehmen und hat dieses dann direkt auf dem Handy zum Download verfügbar. Die Fotobox ist also das perfekte Gadget für Feierlichkeiten. Die Gäste können sich alleine oder in Gruppen vor der Kamera positionieren und dann einfach mit ihrem Handy auslösen.

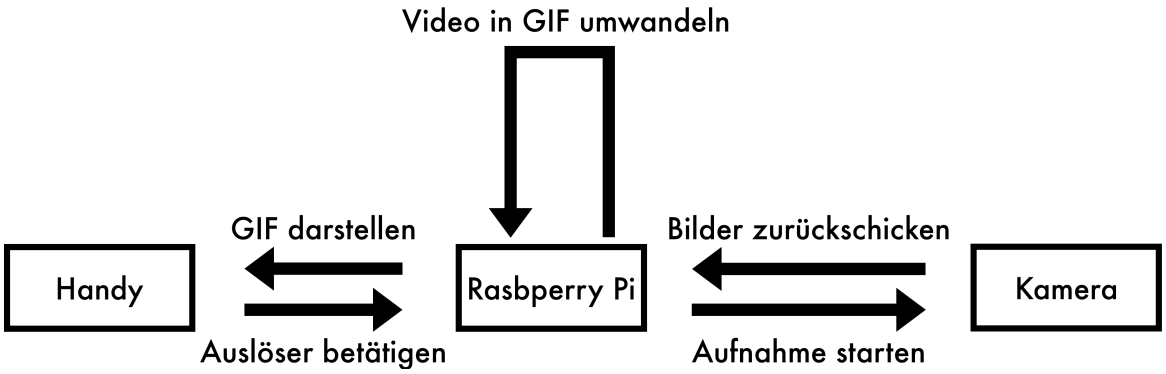
Ablaufdiagramm

Kamera Button im Frontend gedrückt

→ MQTT Befehl an Broker

→ Auf dem Raspberry Pi bei Empfang der Nachricht
- Pi löst Kamera aus (gPhoto2)
- Anwendung von Modus (normal, reverse, boomerang) (FFmpeg)
- Video Umwandlung in Gif (FFmpeg)
- Gif zu base64 (base64-img)
- Veröffentlichung MQTT Nachricht: Gif als base64 Zeichenkette

→ Im Frontend
- Anzeigen des Laderades
- Bei Empfang der Base64 MQTT Nachricht: Gif anzeigen



Genereller Aufbau / Funktionen

Die Frontend Web-App sendet über MQTT Nachrichten an einen Broker. Diese Nachrichten werden ebenfalls von unserem Backend, das sich auf dem Raspberry Pi befindet, empfangen. Alle vom User vorgenommenen Einstellungen für das Erstellen des GIFs (wie zum Beispiel Länge, Bilder pro Sekunde oder auch Farbfilter) werden so nach Betätigen des Auslösebuttons übertragen und dann ausgelesen. Die Kamera wird nun mit Hilfe von GPhoto2 ausgelöst. Dabei wird ihr Liveview-Stream abgegriffen und als MJPG Video auf dem Raspberry Pi gespeichert. Hier wird das Video nun mit Hilfe von FFmpeg bearbeitet. Das Video wird zu einem Gif konvertiert, auf welches bei Auswahl des Nutzers noch Farbfilter (schwarz-weiß) angewendet werden. Um nun auf dem Handy des Users angezeigt werden zu können, muss das GIF per MQTT an den Broker geschickt werden. Hierfür wird es in eine Base64 Zeichenkette umgewandelt.

Front-/ Backend im Detail

Backend

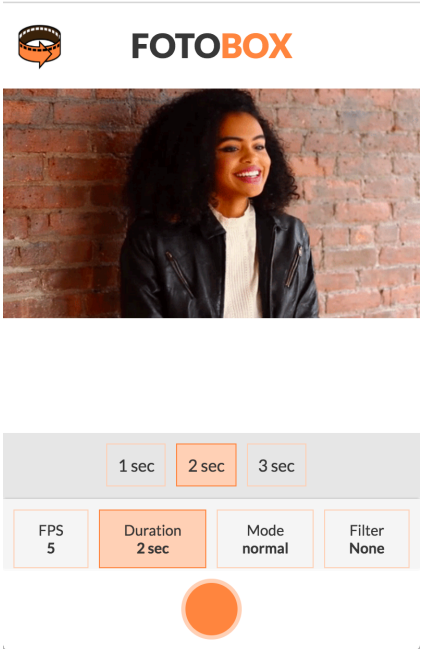
Der Backend-Server auf dem Raspberry Pi ist in Javascript geschrieben und verwendet Node.js. Zusätzlich werden die Bibliotheken gPhoto2 und FFmpeg benutzt. GPhoto2 wird benutzt um eine per USB an den Raspberry Pi angeschlossene Kamera auszulösen und das Video auf dem Pi zu speichern. Mit FFmpeg wird dieses zum Gif konvertiert. Zusätzlich bietet FFmpeg vielfältige Möglichkeiten zur Bearbeitung von Videos, durch die wir die Einstellungsmöglichkeiten der Gifs realisieren.

Bei Serverstart verbindet sich dieser mit dem MQTT Broker. Wird von dem Nutzer der Auslöser betätigt so wird über MQTT der Befehl zum Auslösen übertragen. Dieser beinhaltet die gewählten Optionen des Nutzers, wie Bilder pro Sekunde(fps), die Dauer des Gifs und den gewählten Modus und Filter für dieses.
Anschließend wird mit gPhoto ein Video, je nach gewünschter Länge, aufgenommen. Dieses wird dann in einen Ordner mit zufällig generiertem Namen verschoben, sodass vom nächsten Nutzer nichts überschrieben wird.
Nun werden, je nachdem welchen Modus der Nutzer für sein Gif ausgewählt hat, von FFmpeg verschiedene Operationen durchgeführt. Hat der Nutzer als Modus “Normal” ausgewählt so wird die MJPG-Datei einfach in ein Gif konvertiert. Bei Auswahl der Modi “Reverse” oder “Boomerang” wird das Video zunächst umgekehrt. Bei Ersterem wird aus diesem umgekehrten Video das Gif erstellt und bei Letzterem werden das normale und das umgekehrte Video zusammengefügt und das Resultat zum Gif konvertiert.
Unabhängig vom gewählten Modus wird bei der Konvertierung die vom Nutzer gewählte FPS Einstellung angewendet.
Hat der Nutzer den schwarz-weiß Filter ausgewählt wird dieser nun mit FFmpeg auf das Gif angewendet. Abschließend muss das Gif zum Nutzer zurückgesendet werden. Da auch dies über MQTT realisiert werden sollte, wird das Gif in Base64 codiert und als Zeichenkette über MQTT versendet.

Frontend

Das Frontend läuft über eine Web-App die mit der Javascript Library React geschrieben ist und auf die HiveMQ-Websocket-Client JavaScipt Datei zurückgreift, um die Verbindung zum MQTT Broker aufzubauen.

Beim Aufrufen der Web-App verbindet sie sich mit dem MQTT Broker und zeigt ein zufälliges GIF aus der Giphy API. Der User hat nun die Möglichkeit verschieden Einstellungen vorzunehmen, bevor er die Kamera auslöst. Dazu gehören: FPS, Duration, Mode und Filter.



Beim Auslösen wird dann ein String, mit den Einstellungen des Users, über MQTT an den Server gesendet. Solange das GIF auf dem Raspberry PI erstellt wird, sieht der User einen Loading Spinner. Sobald das GIF fertig ist, wird es anstelle des Loading Spinners angezeigt und ein Download Button wird sichtbar, der das Herunterladen des erstellten GIFs ermöglicht.

Vergleich: Konzept mit wirklicher Umsetzung

Wir konnten die uns in unserem Projektkonzept gestellten Herausforderungen und Anforderungen zum größten Teil erfüllen.
Trotzdem sind wir aufgrund der Benutzerführung und Einfachheit in der Umsetzung an einigen Stellen von unserem eigentlichen Plan abgewichen.
Im ersten Versuch übertrugen wir die gesamten Daten per HTTP. Hierbei war die Übertragung des Bildes um einiges schneller. Wir haben unsere Backend und Frontend App dann allerdings noch so angepasst, das MQTT zur Datenübertragung verwendet wird, da es in den Anforderungen so beschrieben war und es nicht möglich gewesen wäre den Pi als Server über das Internet erreichbar zu machen.
Die Internetseite, die der Nutzer auf seinem Handy aufruft, besteht nun nur noch aus einer Seite. Hier wird am Anfang ein zufällig ausgewähltes Gif dargestellt. Dieses mussten umsetzen, da es keine Möglichkeit gab das Livebild ohne Verzögerung und direkt aus der

Kamera zu übertragen. Hierbei wäre uns vermutlich der MQTT Broker in die Knie gegangen. Deshalb haben wir uns entschieden eine Livebild-Darstellung über einen extra Monitor unterhalb der Kamera zu zeigen. Dieser zeigt ein schwarzes Bild an, sobald die Kamera ausgelöst wurde.
Das zufällig ausgewählte und dem Nutzer gezeigte Gif wird durch das aufgenommene Gif ersetzt, sobald die MQTT Nachricht vom Broker rausgeschickt wurde.
Hierdurch sieht der Nutzer das Gif direkt und muss auf keine zweite Seite weitergeleitet werden. Er kann es von dort aus direkt über den Download Button speichern. Wir haben hierfür also nicht, wie geplant, eine zweite Seite auf die der Nutzer weitergeleitet wird.
Außerdem verwenden wir die Spiegelreflex Kamera nur passiv. Sie sendet keine Daten aktiv, sondern ist lediglich nur eingeschaltet und zeigt ein Vorschaubild die ganze Zeit. Dieses Vorschaubild wird von der Bibliothek GPhoto2 abgegriffen und als Videodatei gespeichert.

Fazit / Lessons learned

Obwohl die Fotobox gut funktioniert, gibt es einige verbesserungswürdige Punkte.
Ein großer Vorteil, um viele Unwägbarkeiten auszumerzen, wäre gewesen, die Kamera anstatt des Liveview-Streams richtige Videos filmen zu lassen, denn es gab häufig Probleme mit der Abspiel-Framerate oder der Videolänge, was mit großer Wahrscheinlichkeit auf Ungenauigkeiten im Abgreifen des Streams zurückzuführen ist.
Eine weitere wichtige Erkenntnis betraf die Wartezeiten für das Pi. Die Konvertierungen und Berechnungen brauchten teilweise, je nach User-Einstellungen, einige Sekunden Zeit, die man dem Pi einräumen und somit auch dem User zumuten musste. Die begrenzte Rechenleistung des Pis war für diese Anwendung gerade so noch ausreichend.
FFmpeg ist ein sehr mächtiges Tool im Bearbeiten von Videos. Es wäre möglich gewesen, viele verschiedene Modifikationen am Video durchzuführen mit vergleichsweise geringem Aufwand, allerdings wäre hier eventuell die Rechenleistung des Pis in die Quere gekommen und hätte für die User ungeeignete Wartezeiten gesorgt. Deshalb haben wir uns entschieden andere Farbfilter wegzulassen.
Die Benutzung von MQTT zum Übertragen von GIFs machte zeitweise Probleme, weil der Maximale Payload durch den Base64-String überschritten wurde (Es entstanden teilweise Zeichenketten mit einer Länge von mehr als 500.000 Zeichen). Eine Übertragung mit einem anderen Protokoll wäre hier vielleicht sinnvoller gewesen, allerdings funktionierte alles nach dem Umstellen auf einen anderen Broker problemlos.
Für die User Experience wäre das Einbinden vom Vorschaubild der Kamera in die App eventuell komfortabler gewesen, vor allem, wenn dieses nach dem Auslösen der Kamera kontinuierlich weitergelaufen wäre (am besten mit einem Symbol für die laufende Aufnahme), da sie so besseres Feedback erhalten hätten, ob die Aufnahme schon gestartet hat, oder nicht und wann sie beendet ist.