**Projekt-Schlussbericht**

**Projekttitel:** Twasi Desktop

**Autoren:** Marco Steiner  
  
 Lars Bärtschi  
  
 Yvo Andri Hofer  
  
 Cyril Heiniger

**Datum:** 08.01.2019

**Inhaltsverzeichnis**

[Teil 1 4](#_Toc534719658)

[1 Aufgabenstellung und Management Summary 5](#_Toc534719659)

[2 Deklaration der Vorkenntnisse 6](#_Toc534719660)

[3 Deklaration der Vorarbeiten und fremden Code 6](#_Toc534719661)

[4 Deklaration der verwendeten Firmenstandards 6](#_Toc534719662)

[5 Zeitplan 7](#_Toc534719663)

[5.1 Phasen Initialisierung und Konzept 7](#_Toc534719664)

[5.2 Phasen Realisierung und Einführung 7](#_Toc534719665)

[6 Arbeitsprotokoll 8](#_Toc534719666)

[6.1 Arbeitsjournal Marco Steiner 8](#_Toc534719667)

[6.2 Arbeitsjournal Lars Bärtschi 9](#_Toc534719668)

[6.3 Arbeitsjournal Yvo Hofer 10](#_Toc534719669)

[6.4 Arbeitsjournal Cyril Heiniger 10](#_Toc534719670)

[Teil 2 12](#_Toc534719671)

[7 Situationsanalyse 13](#_Toc534719672)

[7.1 Ausgangslage 13](#_Toc534719673)

[7.2 Stärken 13](#_Toc534719674)

[7.3 Schwächen 13](#_Toc534719675)

[8 Systemziele 14](#_Toc534719676)

[8.1 Rahmenbedingungen 14](#_Toc534719677)

[8.2 Abgrenzung 14](#_Toc534719678)

[9 Lösungsvorschläge 15](#_Toc534719679)

[9.1 Variantenübersicht 15](#_Toc534719680)

[9.2 Beschreibung der Varianten 15](#_Toc534719681)

[9.3 Bewertung der Varianten (Tabelle) 15](#_Toc534719682)

[9.4 Lösungsbeschreibung 17](#_Toc534719683)

[10 Systemarchitektur 18](#_Toc534719684)

[10.1 Systemdesign 18](#_Toc534719685)

[10.2 Schnittstellendefinitionen 20](#_Toc534719686)

[10.3 Datenmodel 20](#_Toc534719687)

[10.4 Sicherheit 20](#_Toc534719688)

[11 Testkonzept und Testspezifikationen 21](#_Toc534719689)

[11.1 Unit Test 21](#_Toc534719690)

[11.2 Systemtest - Testspezifikation 21](#_Toc534719691)

[11.3 Testprozedur 22](#_Toc534719692)

[12 Einführungskonzept 24](#_Toc534719693)

[12.1 Zusammenfassung 24](#_Toc534719694)

[12.2 Einführungsplan 24](#_Toc534719695)

[12.3 Migrationsplan 24](#_Toc534719696)

[13 Testprotokoll 25](#_Toc534719697)

[13.1 Abnahme 25](#_Toc534719698)

[14 Benutzerdokumentation / Anleitungen 26](#_Toc534719699)

[14.1 Konfigurations-Dokumentation 26](#_Toc534719700)

[14.2 Benutzerhandbuch 26](#_Toc534719701)

[14.3 Integrations- und Installationshandbuch 27](#_Toc534719702)

[14.4 Supporthandbuch 28](#_Toc534719703)

[15 Projekterfahrung 28](#_Toc534719704)

[Teil 3 29](#_Toc534719705)

[16 Dokumentation Quellcode 30](#_Toc534719706)

[17 Literaturverzeichnis 46](#_Toc534719707)

[18 Glossar 46](#_Toc534719708)

**Abbildungsverzeichnis**

[Abbildung 1: Zeitplan Phasen Initialisierung und Konzept 7](#_Toc534719709)

[Abbildung 2: Zeitplan Phasen Realisierung und Einführung 7](#_Toc534719710)

[Abbildung 3: Systemübersicht 18](#_Toc534719711)

[Abbildung 4: Klassendiagramm der Gesamtapplikation 19](#_Toc534719712)

# Teil 1

## Aufgabenstellung und Management Summary

Unser Projekt stellt es sich zum Ziel, den bestehenden Twitch-Bot Twasi um eine Desktop-Applikation zu erweitern, welche dem Benutzer (Streamer auf der Streamplattform Twitch) erweiterte Funktionalitäten bietet und die Arbeit mit dem Twitch-Bot zentralisiert.

Aktuell ist es dem Benutzer nicht möglich die von Twasi gesammelten Daten zu exportieren oder importieren, zudem ist in der aktuell verfügbaren Version von Twasi, eine Interaktion mit Streaming-Software (z. B. OBS) nicht möglich. Die zu entwickelnde Applikation soll eine zentrale Anlaufstelle für den Benutzer darstellen, die auch im Hintergrund verschiedene Aufgaben ausführen oder Daten auswerten kann. Der Twitch-Bot Twasi ist bereits entwickelt, dieser stellt eine API zur Verfügung auf, welche unsere Desktop-Applikation zugreifen kann. Der Twitch-Bot verfügt ebenfalls über eine ausgereifte Web-Applikation, welche durch die zu entwickelnde Applikation nicht ersetzt, sondern erweitert werden soll.

Am Projekt wird während der Unterrichtseinheiten der Module 306 sowie 213, von drei Stunden pro Kalenderwoche gearbeitet.

Für die Phase Initialisierung und die gesamte Projektdauer wird die Projektmethode Hermes angewendet.

Der Code, welcher im Rahmen des Projekts neu erstellt wird, ist an die bisher durch die Entwickler von Twasi umgesetzten Code-Standards anzupassen.

Betreffend dem Mittelbedarf, so ist der Hauptstandort der Entwicklung das Zimmer 207 des Campus IET, hinzu kommt betreffende Arbeit von zu Hause, sollte dies aus Zeitgründen nötig sein. Die physische IT-Infrastruktur zur Entwicklung wird entweder von den Teammitgliedern selber mitgebracht (BYOD), oder von der IET der Berufsschule Gibb zur Verfügung gestellt. Während der Projektzeit steht uns ein Zeitbudget von drei Stunden pro Woche, während 13 Wochen zur Verfügung.

Bei vier Projektmitgliedern resultiert dies über den ganzen Projektzeitraum hinweg (inkl. Projektantrag und Schlussbericht) in 156 Arbeitsstunden.

Bei einem Stundensatz von 20.- würde dies in einem Totalbetrag von 3‘120.- resultieren. Dieser wird jedoch, basierend auf der Tatsache, dass es sich hierbei um ein freiwilliges Projekt der Schule handelt, dessen primäres Ziel es ist, sich mit der Projektplanungsmethode „HERMES“ auseinanderzusetzen, nicht verrechnet.

Betreffend der Organisation, so ist der Auftraggeber zum einen die Klassenlehrperson „Beat Walter“, zum anderen das Mutterprojekt „Twasi.net“. Von den vier Projektmitgliedern schlüpft immer einer in die Rolle des Projektleiters, diese Rolle wird nach jeder abgeschlossenen Projektphase ausgetauscht (Rotation).

Folgende Risiken wurden überprüft und entsprechende Massnahmen formuliert:

Dokumentshare ist nicht mehr erreichbar, dieses Risiko ist als relativ gering zu beurteilen.

Ein Eintritt dieses Falles hätte einen Unterbruch der gemeinsamen Entwicklung zur Folge, was die Mitglieder dazu zwingen würde auf alternative Wege der kollaborativen Zusammenarbeit zurückzugreifen, was eine Leistungsminderung und Verzögerungen bei der Abgabe von Dokumenten zur Folge hätte.

Eine präventive Massnahme wäre hier, häufig benutzte Dokumente lokal abzuspeichern.

Die Server der Versionsverwaltungsplattform GitHub sind nicht mehr erreichbar, dieses Risiko ist als relativ gering zu beurteilen.

Ein Eintritt dieses Falles hätte einen Unterbruch der gemeinsamen Entwicklung zur Folge, was eine Leistungsminderung zur Folge hätte.

Eine präventive Massnahme kann hier nicht getroffen werden.

Krankheit oder längerer Ausfall eines Teammitglieds, dieses Risiko ist als mittelmässig zu beurteilen.

Ein Eintritt dieses Falles hätte eine Einschränkung der Arbeitskapazität zur Folge, welche durch höheren Aufwand der anderen Teammitglieder kompensiert werden müsste.

Eine präventive Massnahme kann hier nicht getroffen werden.

Abwesenheit eines Teammitglieds durch Militär oder anderweitige Gründe höherer Gewalt, dieses Risiko ist als mittelmässig zu beurteilen.

Ein Eintritt dieses Falles hätte eine Einschränkung der Arbeitskapazität zur Folge, welche durch höheren Aufwand der anderen Teammitglieder kompensiert werden müsste.

Eine präventive Massnahme wäre hier, eine Umplanung wichtiger Termine im Voraus.

## Deklaration der Vorkenntnisse

Um das Projekt erfolgreich durchführen zu können, muss Wissen zu den gewählten Technologien vorhanden sein.

Der Variantenentscheid in der Studie (*studie\_twasi-desktop.pdf*) hat die Lösung mit Electron und React hervorgebracht.

Basiskenntnisse in den Bereichen HTML, CSS und JavaScript sind gefordert. Diese Basiskenntnisse sind bereits im ganzen Team in erweitertem Umfang vorhanden.  
Mit der Technologie React war hauptsächlich eine Person vertraut, dieses Wissen wurde durch einen Crash-Kurs zum Beginn der Realisierungsphase geteilt. Kenntnisse im Bereich Electron waren bei Projektbeginn noch nicht vorhanden, da Electron nur einen kleineren Teil bzw. die «Hülle» der Applikation darstellt, stellte dies keinen grossen Problemfaktor dar.

Einige Komponenten aus dem Mutterprojekt «Twasi», die in der «Twasi» Desktop Applikation wiederverwendet werden können waren bereits vorhanden. Diese konnten übernommen werden ohne riesige Anpassungen machen zu müssen. Dort sind auch einige Beispiele, die bei der Entwicklung von Nutzen waren.

Vom Mutterprojekt «Twasi» ist der ungefähre Anmeldevorgang vorhanden, dort sind daher keine spezifischen Kenntnisse von Nöten.

Für die Versionskontrolle zu gewährleisten, wird ein GIT-Repository erstellt. Alle Teammitglieder kennen Git bereits und können damit zur Genüge umgehen.

Als Dokumentablage verwenden wir OneDrive, welches sehr benutzerfreundlich ist und daher auch keine weiteren Kenntnisse Notwendig sind. Zudem können dort Dokumente gemeinsam bearbeitet werden, was zur Produktivität des Teams beiträgt.

Der Code-Editor wird individuell von jedem Mitglied gewählt. So haben dort alle ihre persönliche Präferenz, mit der sie sich auskennen, und niemand wird dazu gezwungen, sich den anderen anzupassen.

## Deklaration der Vorarbeiten und fremden Code

Bereits vorher existiert das Mutterprojekt „Twasi.net“. Erst handelte es sich hierbei um einen Proof of Concept, dieser wurde nach erfolgreicher Validierung eingestellt und so steht das Projekt derzeit in Neuentwicklung. Zur Anbindung an andere Dienste wird eine Schnittstelle angeboten. Als Vorarbeit wurde diese Schnittstelle bereits um die relevanten Endpunkte erweitert und die Funktionalität dieser so weit wie möglich getestet. Endpunkte, die nicht in der geforderten Zeit realisiert werden konnten, wurden durch Mock-Code ersetzt (sie geben Testdaten zurück und lassen sich nach der erfolgreichen Implementation ohne Schwierigkeiten ersetzen).

Weiter wurden bereits diverse Bibliotheken geschrieben, welche Abhängigkeiten des Projektes sind. Die grössten hiervon sind Node.JS (JavaScript Laufzeitumgebung), Atom (Hülle/Brücke für JavaScript-Applikationen zu Desktop-Applikationen) und React (Design & Benutzersteuerungsverwaltung). Vorarbeiten betreffend der Desktop-Applikation selbst sind noch nicht vorhanden, das Projekt deckt die Initialisierung der Applikation ab.

## Deklaration der verwendeten Firmenstandards

Die Entwicklung von Twasi Desktop wurde auf der Basis der Projektmanagementmethode Hermes 5 geplant, auf ihre Machbarkeit geprüft und umgesetzt.

Entsprechend dieser Methode wurden während verschiedenen Projektphasen jeweils Dokumentationen erstellt und die Fortschritte bzw. nächsten geplanten Schritte festgehalten und analysiert.

Die Phasen Initialisierung, Konzept, Realisierung und Einführung waren in diesem Bereich von Bedeutung. Die Anforderungen sowie die daraus resultierenden Arbeiten wurden durch die Einhaltung der Vorgaben zu den Phasen strikt voneinander getrennt, um eine problemlose Entwicklung zu gewährleisten. Als weiterer Standard wurde ein zwischen den Phasen auszutauschender Projektleiter definiert, dieser stellt die Qualitätshaltung sicher und repräsentiert die Entwicklergruppe sowie die gesamte Applikation gegen Aussen und hält sämtliche hier beschriebenen Vorgaben selbst ein und sorgt ebenso für deren Umsetzung.

Insbesondere während der Phase Realisierung, der effektiven Realisierung der Applikation gemäss Planung nach Hermes 5, wurde im Team nach dem agilen Vorgehensmodell Scrum gearbeitet. Typische Elemente der Projektarbeit mit Scrum wie regelmässige Meetings und eine dynamische Arbeitsaufteilung wurden übernommen und während der gesamten Entwicklungszeit von Twasi Desktop eingehalten.

## Zeitplan

### 5.1 Phasen Initialisierung und Konzept



Abbildung 1: Zeitplan Phasen Initialisierung und Konzept

Während der Phasen Initialisierung und Konzept haben wir hauptsächlich an der Ausarbeitung der Dokumente «Studie», «Projektauftrag» sowie «Konzeptbericht» gearbeitet. Diese Arbeit an den Dokumenten der Phase Initialisierung hat das Arbeitspensum gefüllt, daher steht die «Soll» der «Ist»-Zeit ausgeglichen gegenüber.

Während der Phase Konzept steht die «Ist»-Zeit dem «Soll»-Pensum etwas nach, dies ist mit der gleichzeitigen Abarbeitung des ersten Dokumentes betreffend der Teamreflektion zu begründen, welches aufgrund dessen Eigenschaft als projektfremdes Zeitobjekt nicht im Excel vermerkt wurde.

So wurde während den beiden Kalenderwochen 38 und 42 pro Person zwei Stunden für diese Dokumente aufgewendet.

So können die fehlenden acht «Soll»-Stunden einfach erklärt werden.

### 5.2 Phasen Realisierung und Einführung

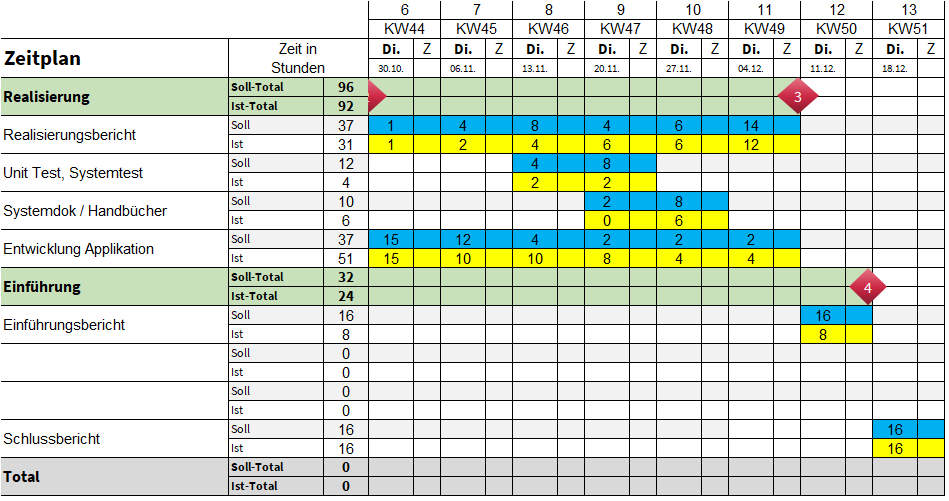


Abbildung 2: Zeitplan Phasen Realisierung und Einführung

Während der Phase Realisierung, der längsten und der gleichzeitig auch vielseitigsten Phase hat sich das Projektteam mit vielen Aufgaben auf kurzer Zeitbasis beschäftigt. Durch eine effiziente Zeitplanung und Arbeitsaufteilung während der Arbeit am Realisierungsbericht konnten wir diesen mit weniger «Ist» als «Soll»-Stunden abschliessen, während in die Entwicklung der Applikation mehr Zeit investiert wurde.

Die Arbeiten an der Systemdokumentation, den Systemhandbüchern sowie den Systemtests konnten wir in relativ kurzer abschliessen, insbesondere die Arbeiten in den Kalenderwochen 46 bis und mit 48 erforderten ein gutes Zeitmanagement und eine effiziente Abarbeitung der benötigten Tasks.

Während der Phase Einführung wurde ebenfalls eine Arbeitsbeteiligung an einer Teamreflektion ausgeführt, was das Arbeitspensum im ersten Teil halbierte. Der Schlussbericht wurde wieder mit vollem Arbeitsaufwand ausgeführt.

Krankheiten oder ähnliche Ausfälle kamen während der gesamten Projektzeit lediglich ein Mal vor, diese traten somit nicht ins Gewicht und können in Anbetracht der erzielten Endleistung

Während den Phasen Realisierung und Einführung wurde in einem Abstand von jeweils drei bis vier Wochen je an einem Dokument zur Teamreflektion gearbeitet, welche hier aus obenstehenden Gründen nicht näher vermerkt sind.

## Arbeitsprotokoll

### 6.1 Arbeitsjournal Marco Steiner

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Marco Steiner, INF2016.5E, 27.11.2018***  ***Tätigkeiten*** | ***Beteiligte***  ***Personen*** | | ***Aufwand***  ***geplant (Std)*** | ***Aufwand***  ***effektiv (Std)*** |
| Einführung Präsentation | Ganze Klasse mit Lehrperson | | 0.1 | 0.1 |
| Daily Scrum Meeting | Projektteam Twasi-Desktop | | 0.2 | 0.2 |
| Arbeit an Übersichtsseite von Twasi-Desktop | Marco Steiner | | 1.4 | 1.6 |
| Klärung Unklarheit React | Lars Bärtschi, Marco Steiner | | 0.3 | 0.3 |
| Aktualisieren Projektplan | Marco Steiner | | 0.3 | 0.4 |
| Lernjournal schreiben | Marco Steiner | | 0.6 | 0.7 |
| Senden der Dokumente an Lehrperson | Marco Steiner | | 0.2 | 0.2 |
| Total: |  | | **3.1** | **3.5** |
| Tages Ablauf | |  |  |  |
| Heute gab es wie immer eine Einführung der Lehrperson in das heutige Thema. Das heutige Thema ist weiterhin der Realisierungsbericht, spezifisch die Systemdokumentation mit Benutzer- und Systemhandbuch, Integrations- und Installationshandbuch.  Danach haben wir unser Daily Scrum Meeting durchgeführt. Wir haben unseren Fortschritt ausgetauscht und einzelne Tasks als fertig markiert. Nach dem Meeting ist mir aufgefallen, dass ich vergessen habe, eine Frage zu stellen. Ich wollte wissen, wo die anderen die Konfiguration einer Seite speichern würden. Zum Glück konnte ich dies auch noch nach dem Meeting klären.  Anschliessend habe ich mich weiter an meine Arbeit gemacht. Ich bin dran, die Übersichtsseite zu erstellen. Dazu habe ich letztes Mal eine Ansicht mit festen werten erstellt. Das Ziel von heute war es, diese durch reale Werte zu ersetzen. So habe ich angefangen, eine Funktion zu erstellen, welche möglichst variable Werte annimmt, die Werte gemäss einer Konfiguration abfragt. Dies ist sehr gut gelaufen. Ich konnte nach ungefähr einer halben Stunde diese Werte sehen. Nun sollen diese noch in das User-Interface gelangen. Ich dachte, dass dies dann sehr einfach wird, doch ich habe mich geirrt. Ich hatte damit Probleme, da ich nicht wusste, wie React auf asynchrone Daten reagiert. Dies versuchte ich alleine zu lösen, was leider nicht klappte. Um schnell weiter zu kommen, habe ich bei Lars Bärtschi nachgefragt, wie dies funktionieren könnte. Da er schon wesentlich mehr Erfahrung mit React hat, konnte er mir sofort helfen. Er nannte das Stichwort «State». Er hat mir erklärt, wie ich in React mit dem «State» umgehen kann. Durch diese Hilfestellung konnte ich dann auch weiterfahren. Die Übersicht sollte nun zwei Verschiedene Werte anzeigen, doch erscheinen tut nur einer. Ich habe nach 5 Minuten Ahnungslosigkeit herausgefunden, dass ich den «State» zu früh kopiert habe und es somit einen Fehler gab.  Nach der Behebung dieses Fehlers war ich nach eineinhalb Stunden mit der Übersichtsseite fertig.  Als Projektleiter habe ich zudem noch die Verpflichtung den Projektplan stets zu aktualisieren und an die Lehrperson zu verschicken. Daher habe ich diesen bearbeitet und versendet. | | | | |
| Hilfestellungen | |  |  |  |
| Ich habe mich heute weiter mit React befasst. Da ich einfachere Funktionen begriffen habe, musste ich nur schwerere Konzepte recherchieren. Dazu gehörte die Darstellung von Daten, die erst später in das GUI geladen werden sollen. Ich war dabei im Internet erfolglos. Ich habe da Konzept nicht ganz verstanden und somit habe ich mich dazu entschieden, Lars Bärtschi direkt zu fragen, da er React bereits gut kennt. Er konnte mir dann die Problemstellung erklären und nun weiss ich darüber auch Bescheid. | | | | |
| Reflexion | |  |  |  |
| Heute verlief das Daily Scrum Meeting sehr gut. Alle konnten erklären, was sie machen und was noch ansteht. Ich habe leider vergessen eine Frage zu stellen, aber dies war auch nicht weiter schlimm, da ich sie später am Arbeitsplatz noch klären konnte.  Die heute behandelten Themen waren sehr interessant und somit war ich auch motiviert daran zu arbeiten. Daher konnte ich mich heute gut auf meine Arbeit fokussieren. Das Arbeitsklima war nicht immer optimal, da es im Klassenzimmer manchmal recht laut ist, aber ich bin mir das mittlerweile gewohnt, daher hat mich das nicht gross gestört.  Ich habe aber auch noch andere Verpflichtungen, wie an Dokumenten zu schreiben. Da bin ich mich noch am Üben. Ich bin meist recht lang am Reflektieren, da ich sichergehen will, dass es in einer guten Qualität ist.  Meine Erkenntnis von heute ist, dass ich Dokumente jeweils frühzeitig schreibe und dass sich so kein Stress entwickelt.  Ich habe heute weitere Spezialitäten von React kennengelernt und kann diese bei einem nächsten Mal direkt anwenden. | | | | |
| Nächste Schritte | |  |  |  |
| Nächste Woche sollten noch die ganzen Dokumente fertiggestellt werden. Zudem sollte noch eine bessere Fehlerbehandlung eingeführt werden, um Abstürze zu verhindern. | | | | |

### 6.2 Arbeitsjournal Lars Bärtschi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Lars, Bärtschi, INF2016.5E, 06.11.2018***  ***Tätigkeiten*** | ***Beteiligte***  ***Personen*** | | ***Aufwand***  ***geplant (Std)*** | ***Aufwand***  ***effektiv (Std)*** |
| Daily Scrum | Projektmitarbeiter (Marco Steiner, Yvo Hofer, Cyril Heiniger),  Lars Bärtschi | | 0.25 | 0.15 |
| Teamrollen: Reflexion schreiben | Lars Bärtschi | | 0.75 | 0.75 |
| Implementation: Wrapper API (vereinfachter Zugriff) | Projektmitarbeiter | | 1.5 | 1.5 |
| Ausfüllen Arbeitsjournal | Lars Bärtschi | | 0.5 | 0.5 |
| Tages Ablauf | |  |  |  |
| An der Applikation wurde das erste Mal wirklich etwas entwickelt. Zwar geht es derzeit noch um architektonische Sachen (wie zum Beispiel den API-Wrapper, den ich implementiert habe). Dabei konnte ich einen ersten Durchstich auf die API erreichen. So konnten wir z.B. die E-Mail-Adresse des aktuell angemeldeten Benutzers mit Hilfe der API auslesen. Somit bin ich mit dem Vorankommen sehr zufrieden. | | | | |
| Hilfestellungen | |  |  |  |
| Hilfestellungen zu React generell: <https://reactjs.org/docs>Hilfestellungen zu Electron (native API): <https://electronjs.org/docs>Hilfestellungen zu JavaScript: <https://developer.mozilla.org/de/docs/Web/JavaScript>Hinweise zu spezifischeren Anwendungsfällen: <https://stackoverflow.com/> | | | | |
| Reflexion | |  |  |  |
| Durch das schnelle Konsolidieren der jeweiligen Dokumentation, konnte Zeit beim Probieren (Trial & Error) gespart werden. Das war sehr gut und deshalb behalte ich das am besten so bei.  Auch sehr gut war die kollaborative Arbeitsweise, sodass wir uns bei Problemen oder Schwierigkeiten weitergeholfen und uns gegenseitig beraten haben. Dadurch konnten wir die Qualität unserer Entscheide untermauern.  Generell bin ich sehr mit der heutigen Arbeit zufrieden und nehme dies für das nächste mal mit. Weiter würde ich häufiger mit Git Zwischenfortschritte speichern (Committen) damit die Versionierung einfacher fällt. | | | | |
| Nächste Schritte | |  |  |  |
| Task APIConnector abgeschlossen, nächster Task: Ausloggen/Abmelden. Damit werde ich das nächste Mal beginnen. | | | | |

### 6.3 Arbeitsjournal Yvo Hofer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Yvo Hofer, INF2016.5E, 13.11.2018***  ***Tätigkeiten*** | ***Beteiligte***  ***Personen*** | | ***Aufwand***  ***geplant (Std)*** | ***Aufwand***  ***effektiv (Std)*** |
| Daily Scrum Meeting | Marco Steiner, Cyril Heiniger, Yvo Hofer, Lars Bärtschi | | 0.16 | 0.16 |
| Update OneDrive mit Realisierungsbericht, Archivieren von älteren Dokumenten | Yvo Hofer | | 0.08 | 0.16 |
| Erstmaliges Ausfüllen des Realisierungsberichts | Yvo Hofer | | 0.5 | 0.3 |
| Implementation Datenexport in Twasi Desktop | Lars Bärtschi, Yvo Hofer | | 1.51 | 1.63 |
| Ausfüllen des Arbeitsjournals | Yvo Hofer | | 0.75 | 0.75 |
| **Total:** |  | | **3** | **3** |
| Tages Ablauf | |  |  |  |
| Ich habe zu Beginn des heutigen Tages habe ich während des Daily Scrum meine Teammitglieder über meine erledigten Tätigkeiten während des letzten Arbeitstages informiert und den neuen Task zur Implementation der Möglichkeit eines Datenexports übernommen. Zusätzlich habe ich mich im Team abgesprochen und den Realisierungsbericht erstmalig geöffnet, angepasst und dem Team auf unserer Teamablage (OneDrive) zur Verfügung gestellt. Auf erwähnter Ablage habe ich ältere Dokumente in das Archiv verschoben um den Überblick behalten zu können.  Gemeinsam mit Lars Bärtschi, welcher das Twasi Projekt schon längere Zeit kennt, habe ich schliesslich mit der Implementation meiner zugeteilten Aufgabe aus dem Daily begonnen, ich habe mir hierzu die bereits bestehende API angeschaut und die neue Funktionalität implementiert, dabei habe ich darauf geachtet bestehende Codevorgaben einzuhalten (Clean Code) und konnte einen auftretenden Authentifizierungsfehler bei Twasi gemeinsam mit Lars Bärtschi beheben.  Ich habe mich damit befasst, bestehende Befehle, mit welchen der Twasi-Bot gesteuert werden kann, in ein für den User praktisch zu lesendes CSV-File zu exportieren, diese Funktionalität konnte ich erfolgreich im gegebenen Zeitrahmen implementieren. | | | | |
| Hilfestellungen | |  |  |  |
| Während der Umsetzung der Arbeiten habe ich mich im Internet (Stackoverflow, reactjs.org) bei Problemen mit React und dessen Syntax umgesehen und konnte stets innert kurzer Zeit eine Lösung finden, beispielsweise für das Auftrennen von Strings in React konnte ich zusätzlich auf bestehendes Wissen im Bereicht JavaScript zurückgreifen, welches ich auch im Betrieb anwende.  Bei fachlichen Problemen mit Twasi habe ich mich direkt bei Lars Bärtschi gemeldet, welcher die Probleme stets rasch lösen konnte. So hatte ich beispielweise ein Authentifikationsproblem bei der API welches auf einen abgelaufenen Authentifizierungstoken zurückzuführen war, diesen konnte ich via Lars Bärtschi abgleichen und mit seiner Hilfe selbst einen neuen Token anfordern, so konnte ich erneut Zeit einsparen. | | | | |
| Reflexion | |  |  |  |
| Die Zusammenarbeit im Team hat durch den heutigen Tag hindurch gut funktioniert.  Während der Arbeit an der Neuimplentation des Export-Features konnte ich auf die Hilfe des Teams zählen und hatte auch das nötige Rüstzeug mich im Internet selbst über Probleme und Fragestellungen zu informieren, ich konnte so mein persönliches Ziel, die Implementation eines Hauptfeatures erreichen und werde meine Arbeitsmethodik so weiterführen.  Ich hatte und habe nun die Möglichkeit mein Wissen im Bereich React-Programmierung auszubauen, mit dem klaren Ziel eine Wissensquelle für unser Team darzustellen und die Teammitglieder in Fragen zu unterstützen. | | | | |
| Nächste Schritte | |  |  |  |
| Im Blick auf das stets näherkommende Ende der Realisierungsphase werde ich mich auch in der kommenden Woche auf die effiziente Umsetzung von offenen Tasks (Backlog) einsetzen und diese unter Bezug der oben erwähnten Informationsquellen ohne Verzögerung zu implementieren versuchen.  Ein wichtiges Kernfeature wird die Implentation des Datenimports von Befehlen und Musikwünschen sein, mit welcher ich mich noch befassen werde. | | | | |

### 6.4 Arbeitsjournal Cyril Heiniger

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Cyril Heiniger, INF2016.5e, 20.11.2018***  ***Tätigkeiten*** | ***Beteiligte***  ***Personen*** | | ***Aufwand***  ***geplant (Std)*** | ***Aufwand***  ***effektiv (Std)*** |
| Einführung von Herrn Walter | Herr Walter, die ganze Klasse | | 0.10 | 0.10 |
| Daily Scrum in den Teams | Lars Bärtschi, Marco Steiner, Yvo Hofer und ich | | 0.10 | 0.25 |
| Implementation von UI für den Export der Songrequests | ich | | 1.70 | 1.55 |
| Input von Herrn Walter | Herr Walter, die ganze Klasse | | 0.10 | 0.10 |
| Arbeitsjournal | ich | | 1.00 | 1.00 |
| Total: |  | | **3.00** | **3.00** |
| Tages Ablauf | |  |  |  |
| Die Einführung von Herrn Walter verlief gut, wie immer. Ich mag diese Art von Einführung, da sie nicht allzu lang gehen, die Information kurz und knackig, aber trotzdem verständlich übermittelt werden. Jedoch hatte ich gegen Ende der Präsentation etwas mit der Konzentration zu kämpfen, zum Glück war dies nur am Ende, so habe ich nicht zu viele Informationen verpasst.  Das Daily war in meinen Augen gut verlaufen, ich habe einen neuen Task übernommen, die Implementierung des UIs für den Export der Songrequests. Da hatte ich erst Mühe reinzukommen, da ich bei der Verwendung von meinem Element mit HTML zu kämpfen hatte. Mit der Hilfe von meinen Teammitgliedern kam ich dann trotzdem auf die Sprünge und habe den ersten Entwurf gepusht, sodass Yvo am Songexport arbeiten konnte. Hier merkte ich, dass es mir guttut, wieder einmal an einer Frontend Applikation zu arbeiten, da das nicht meine grösste Stärke ist.  Nach der Pause zeigte uns Herrn Walter ein Video zur Teamkommunikation, was sicher ein wichtiger Punkt ist, jedoch in meinen Augen in unserem Team gut gehandhabt wird.    Danach nahm ich mir die Zeit, um das Arbeitsjournal zu machen, damit ich dann gegen Ende des Unterrichts nicht in den Stress komme. Hier kam ich schnell ins Schreiben und kam auch gut voran, wodurch ich danach weiter am Design von meiner Komponente arbeiten konnte. | | | | |
| Hilfestellungen | |  |  |  |
| Hilfestellung von Yvo Hofer und Lars Bärtschi zu HTML und React Electron.  [www.material-ui.com](http://www.material-ui.com/) Hilfestellung für React Electron Material-UI Library. | | | | |
| Reflexion | |  |  |  |
| In meinen Augen verlief das Daily ziemlich gut, da ich mir heute einen interessanten, trotzdem lösbaren Task nehmen konnte und mit diesem gut vorankam.   Mit dem Arbeitsjournal kam ich auch gut voran, sodass ich dieses bis zum Lektionsende abschliessen konnte.  Beim Mergen habe ich etwas vergessen raus zu löschen, dieses musste dann Yvo rauslöschen.  Das Ziel ist, dass sowas nicht mehr vorkommt, dass ich in Zukunft sauber die Sachen merge und diese Aufgabe nicht von anderen Teammitgliedern vorgenommen werden muss. | | | | |
| Nächste Schritte | |  |  |  |
| Der nächste Schritt, ist das Beenden von meinem Design, dass ich den nächsten Task übernehmen kann und im Team immer wie produktiver mitarbeiten kann. | | | | |

# Teil 2

## Situationsanalyse

### 7.1 Ausgangslage

Die Grundlage unseres Projekts stellt der Twitch-Bot Twasi dar. Dieser verfügt über eine Backend-API sowie ein Webpanel.

Mit dem Twitch-Bot Twasi lässt sich der Chat des Live-Streaming-Videoportals Twitch moderieren; Benutzer können verschiedene Anfragen an den Bot stellen, worauf dieser mit einer Textnachricht antwortet, es können Gewinnspiele gestartet und ausgewertet werden, dies entlastet den Streamer.

Das Twasi-Webpanel bietet dem Benutzer die Möglichkeit, die Aktivität des Twitch-Bots zu überwachen sowie Befehle und Plugins ein- und auszuschalten.

Für die Betaphase des Twitch-Bot Twasi haben sich über 2'000 Nutzer registriert, wovon aktuell rund 80 Personen aktive Benutzer sind, Tendenz steigend.

### 7.2 Stärken

|  |  |
| --- | --- |
| Nr | Beschreibung |
| 01 | Das Twasi-Webpanel bietet bereits viele Funktionen, welche nicht ersetzt werden sollen, da diese an das Webpanel angepasst sind. |
| 02 | Das Design von Twasi (z.B. Webpanel) ist bereits sehr übersichtlich und minimalistisch gestaltet. Dies sollte sich auch in der Desktop-Applikation wiederspiegeln. |
| 03 | Der Twitch-Bot Twasi läuft aktuell sehr stabil und ohne Unterbrüche.  Die Antwortzeiten der API sind sehr schnell, dies ist wünschenswert. |

### 7.3 Schwächen

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | Beschreibung | Beurteilung | Ursache |
| S01 | Um eine Übersicht vom Status von Twasi zu haben, muss die Webseite geöffnet sein. | Funktioniert je nach Webbrowser mehr oder weniger gut, zudem ist es nicht eigenständig. | Fehlende Desktop-Applikation eines bestehenden Dienstes. |
| S02 | Eine Integration in das lokale System wird nicht geboten, zum Beispiel Datenimporte oder Exporte ins oder aus dem Dateisystem. | Kann derzeit nicht angeboten werden, da die Steuerung über ein Webinterface läuft und dazu native API’s verwendet werden. | Fehlende Desktop-Applikation eines bestehenden Dienstes. |

## 

## Systemziele

Die nachfolgende Tabelle gibt, tabellarisch sortiert, einen geordneten Einblick in die systemrelevanten Ziele von Twasi Desktop, wobei «Z» für «Ziel» steht.

|  |  |
| --- | --- |
| Nr | Beschreibung |
| Z01 | Die Twasi-Desktop Applikation soll eine zentrale Anlaufstelle für den Benutzer bieten. Dabei soll eine praktische Übersicht dargestellt werden, so wie Quicklinks zu den entsprechenden Optionen im Web-Panel. |
| Z02 | Über erweiterte Einstellungen können Daten aus der entfernten Datenbank lokal abgespeichert werden. Zum Beispiel das aktuell abgespielte Lied oder Informationen zum aktuell laufenden Gewinnspiel. Diese können dann zum Beispiel wiederum von der Aufnahmesoftware als Textquelle eingefügt werden. Auch besteht ein lokales Backup der entsprechenden Daten, dass auch selbst modifiziert werden kann (zum Beispiel durch das Erstellen von Scripts). Die Daten werden nach einem verbreiteten und bekannten Standard gespeichert. |
| Z03 | Songrequests, so genannte Liedwünsche der Zuschauer, können über die Desktop-Applikation direkt im Hintergrund abgespielt werden. Zudem soll so die Lautstärke unabhängig der Browserlautstärke anpassbar sein. |
| Z04 | Der Streamtitel (Überschrift auf dem Live-Streaming-Videoportal Twitch.tv), wie auch das gespielte Spiel (Metadaten zum Livestream) können direkt auf der Übersichtsseite mit nur einem Klick geändert werden. Dies spart das Öffnen des Webinterfaces des Anbieters Twitch.tv. |
| Z05 | Auf der Übersichtsseite werden zudem auch Echtzeit-Informationen zum aktuellen Livestream eingeblendet. Dazu zählen zum Beispiel die Zuschauerzahl oder der Verlauf der Zuschauer, zum Beispiel als Diagramm dargestellt. |

### 8.1 Rahmenbedingungen

Am Projekt wird während der Unterrichtseinheiten der Module 306 sowie 213, von drei Stunden pro Kalenderwoche gearbeitet. Dies während der gesamten Modulzeit, sprich während 6 Monaten. Ausgenommen Schulferien.

Für die Phase Initialisierung und die gesamte Projektdauer wird die Projektmethode Hermes angewendet. Der Quellcode, welcher im Rahmen des Projekts neu erstellt wird, ist an die bisher durch die Entwickler von Twasi umgesetzten Code-Standards anzupassen.

Die Entwicklung wird hauptsächlich auf der Infrastruktur der Gibb betrieben (IDE, Programmierung). Das Hosting und das Bereitstellen der API geschieht durch Twasi.net (Twasi-Team).

### 8.2 Abgrenzung

Das Projekt Twasi-Desktop Applikation besteht lediglich aus der Entwicklung der Twasi-Desktop Applikation. Weiter gehört zum Projektabschluss die initiale Inbetriebnahme und das Testing der Applikation auf verschiedenen verbreiteten Systemen.

**Nicht** Projektbestandteil sind folgende Punkte:

* Hosting & zur Verfügung stellen der Schnittstelle zu Twasi (API)
* Nötige Erweiterung der Schnittstelle zu Twasi (API)
* Betrieb & Support

## Lösungsvorschläge

### 9.1 Variantenübersicht

Für die Umsetzung der definierten Anforderungen sind unterschiedliche Lösungsvarianten denkbar, in welchen jeweils die verwendete Programmiersprache und gewählte Entwicklungsumgebung variiert.

Wir haben uns hierzu von einer bereits vorhandenen Lösungsidee distanziert und, zusätzlich zu dieser, zwei weitere Varianten in die Entscheidung einbezogen.

*Variante a)*

React und Electron; plattformunabhängige Desktop-Applikation auf Web-Basis.

*Variante b)*

C# und WPF; Desktop-Applikation im Microsoft Windows-Stil.

*Variante c)*

Java und JavaFX; plattformunabhängige Desktop-Applikation.

### 9.2 Beschreibung der Varianten

*Variante a)*

In Variante a) ist das Ziel, eine plattformunabhängige Desktop-Applikation auf Basis des Electron-Frameworks zu erstellen, diese würde mit JavaScript und dessen Bibliothek React realisiert werden.

Einige Komponenten aus dem bestehenden Twasi-Webpanel können hier weiterverwendet werden, zudem lässt sich das bestehende Know-how zur Entwicklung von Webapplikationen hier zielführend einbringen.

Das Framework React läuft unter dem Motto: «Learn once, use everywhere».

Die Applikation wird unter denselben Vorgaben und Anforderungen entwickelt wie eine normale Website, mit der Hilfe des Electron-Frameworks kann die Applikation aber wie bekannte Desktop-Applikationen gestartet und benutzt (Dateizugriffe / Windows-Geräte) werden, ohne dass der Benutzer einen Browser oder dergleichen benutzen muss.

*Variante b)*

In Variante b) ist das Ziel, eine Desktop-Applikation auf Basis der Microsoft Windows Presentation Foundation zu erstellen, diese würde mit C# realisiert werden.

Komponenten des Twasi-Webpanels könnten hier nicht weiterverwendet werden, da hier eine völlig neue Umgebung zum Einsatz kommen würde. Know-how zur Entwicklung von Applikationen mit WPF ist vorhanden und könnte zielführend eingebracht werden.

Die Applikation wird mit XAML gestaltet und die Daten und Aktionen mit C# programmiert, eine solche Applikation kann auf Dateien und Windows-Geräte zugreifen.

Diese Variante ist nicht plattformübergreifend und müsste für den Einsatz auf anderen Betriebssystemen abgepasst werden.

*Variante c)*

In Variante c) ist das Ziel, eine plattformunabhängige Desktop-Applikation auf Basis der JavaFX zu erstellen, diese würde mit Java realisiert werden.

Komponenten des Twasi-Webpanels könnten hier nicht weiterverwendet werden, da hier eine völlig neue Umgebung zum Einsatz kommen würde. Know-how zur Entwicklung von Applikationen mit JavaFX ist nicht vorhanden und könnte nicht eingebracht werden.

Die Applikation wird mit JavaFX und die Daten und Aktionen mit Java programmiert, eine solche Applikation kann auf Dateien und Windows-Geräte zugreifen.

### 9.3 Bewertung der Varianten (Tabelle)

*Variante a)*

| Kriterium | Gewicht | Bewertung | Total |  |
| --- | --- | --- | --- | --- |
| Abdeckung der Anforderungen | 25% | 3 | 0.75 | Die Applikation mit Electron und React erfüllt die gestellten Anforderungen umfassend. |
| Realisierbarkeit, Risiken | 25% | 3 | 0.75 | Die Umsetzung mit Electron und React umfasst minimale Risiken, der Aufwand ist übersichtlich. |
| Wirtschaftlichkeit | 20% | 3 | 0.6 | Der Entwicklungsaufwand ist dank der Verwendung bereits bekannter Technologien gering, die Applikation gehört zu einem bereits rege benutzten Ökosystem. Electron liefert effiziente Werkzeuge zum Erstellen von Installationsdateien. |
| Wartbarkeit | 20% | 2 | 0.4 | Die Einführung neuer Entwickler dauert, dank dem Einsatz neuer Technologien, wenig Aufwand, benötigt aber kleine Anpassung Seiten des Entwicklers (React). |
| Plattformunabhängigkeit | 10% | 3 | 0.3 | Die Applikation ist plattformunabhängig. Die Electron-Community liefert kontinuierlich Updates. |
| **Gesamtbeurteilung** | 100% |  | **2.8** |  |

*Variante b)*

| Kriterium | Gewicht | Punkte | Total |  |
| --- | --- | --- | --- | --- |
| Abdeckung der Anforderungen | 25% | 2 | 0.5 | Die Applikation mit WPF und C# erfüllt die gestellten Anforderungen, bis auf die Plattformunabhängigkeit. |
| Realisierbarkeit, Risiken | 25% | 2 | 0.5 | Die Umsetzung mit Electron und React umfasst minimale Risiken, der Aufwand ist übersichtlich.  Beim Entwickeln der Applikation könnten z.B. bei der Implementation von Funktionen, zu welchen wir nur wenig Kenntnisse haben Probleme auftreten. |
| Wirtschaftlichkeit | 20% | 2 | 0.4 | Der Entwicklungsaufwand ist dank der Verwendung bereits bekannter Technologien gering, jedoch kann nicht garantiert werden, dass jede gestellte Anforderung mit C# umgesetzt werden kann. WPF unterliegt Microsoft, so gibt es wenige Updates seitens der Community. |
| Wartbarkeit | 20% | 2 | 0.4 | Die Einführung neuer Entwickler erfordert, dank dem Einsatz neuer Technologien, wenig Aufwand. Die Tools zur Wartung der Applikation stammen von Microsoft und bedürfen geschulten Entwicklern. |
| Plattformunabhängigkeit | 10% | 0 | 0 | Die Applikation benötigt ein Windows-Hostsystem. |
| **Gesamtbeurteilung** | 100% |  | 1.8 |  |

*Variante c)*

| Kriterium | Gewicht | Punkte | Total |  |
| --- | --- | --- | --- | --- |
| Abdeckung der Anforderungen | 25% | 2 | 0.5 | Die Applikation mit JavaFX unterstützt die gestellten Anforderungen, bietet allerdings keine Vorteile bezüglich Wiederverwendbarkeit der Komponenten. |
| Realisierbarkeit, Risiken | 25% | 1 | 0.25 | Es gibt einige Risiken bei der Entwicklung mit Java, zum Beispiel die eigene Code-Signierung und den eigenen Installer. Zudem weist Java zum Teil gravierende Sicherheitslücken auf, die ein regelmässiges Updaten des Benutzers erfordern. |
| Wirtschaftlichkeit | 20% | 1 | 0.2 | Die Applikation ist abhängig von Java, erzwingt daher die Installation auf dem Computer des Benutzers. Dabei ist Java sehr Ressourcenlastig und weist beim unregelmässigen Updaten Sicherheitsfehler auf. |
| Wartbarkeit | 20% | 3 | 0.6 | Die Java-Applikation weist sich als sehr wartbar aus, da bereits intern bei Twasi vielerorts Java eingesetzt wird und es daher nicht an Entwicklern mangeln würde. |
| Plattformunabhängigkeit | 10% | 2 | 0.2 | Die Java-Applikation ist so Plattformunabhängig wie Java es ist. Läuft zwar auf vielen Umgebungen, erhält allerdings eher selten Unterstützung von bestehenden Nutzern, um neue Plattformen hinzuzufügen. Welche Plattformen unterstützt werden wird vom Inhaber, Oracle, bestimmt. |
| **Gesamtbeurteilung** | 100% |  | 1.75 |  |

Bewertung: 1-3, Gewichtung in %

### 9.4 Lösungsbeschreibung

Da vieles rund um Twasi bereits besteht und Twasi selbst auch eine einheitliche API anbietet (GraphQL), kann die Desktop-Applikation voll und ganz von diesen profitieren.

Durch obenstehende Analyse der Varianten haben wir uns für Variante a) mit Electron und React entschieden.

Da auch einige Funktionen, die die Applikation anbieten soll, bereits heute im Web-Panel verfügbar sind, können diese Komponenten auch von dort wiederverwendet werden.

Weiter haben wir dank Electron nicht Mühe, uns um Dinge wie die Installation der Applikation, manuelle UI-Gestaltung (Swing/WPF) oder die unterschiedlichen Ansprechmethoden der nativen APIs (Filesystem, Lautstärke) zu kümmern.

## Systemarchitektur

### 10.1 Systemdesign

#### 10.1.1 Struktur des Systemdesigns

Der Benutzer von Twasi Desktop hat die Möglichkeit, Daten über einen Direktzugriff aus der Applikation in sein lokales Dateisystem zu exportieren.

Während der Nutzung von Komponenten, welche einen Serverzugriff benötigen, kommuniziert die Applikation via GraphQL mit dem Twasi.net Server, dieser wiederum kommuniziert mit der Twitch.tv API und zeigt deren Daten.

Twasi.net verfügt weiter über eine Datenbank zur direkten Speicherung von Nutzerdaten. (Caching)

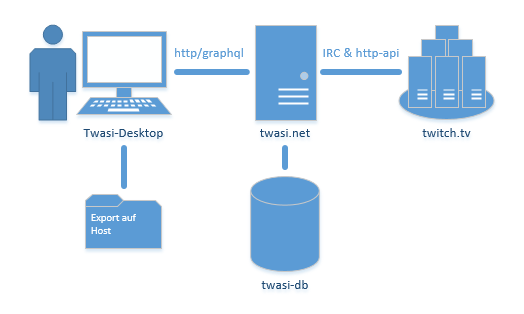


Abbildung 3: Systemübersicht

#### 10.1.2 Beschreibung der Elemente

*Twasi-Desktop*

Bei Twasi-Desktop handelt es sich um die Applikation, die in diesem Projekt entwickelt wird. Die Applikation erlaubt es, Dateien auf das Dateisystem zu exportieren.

*Twasi.net*

Twasi.net ist das Mutterprojekt von Twasi Desktop und ist derzeit auch noch in der Realisierung. Es stellt eine http-Schnittstelle zur Verfügung, über welche Twasi Desktop Daten abfragen kann. Weiter ist es an eine Datenbank angebunden, die für das Zwischenspeichern und ergänzen von Informationen verwendet wird. Twasi.net stellt auch die Schnittstelle zum Streaming-Anbieter Twitch.tv her.

*Twitch.tv*

Twitch.tv ist ein Streaminganbieter, welcher mit dem Projekt direkt keine tiefere Beziehung hat. Über Twasi.net spricht Twasi Desktop zu Twitch und fragt Daten ab. Twitch bietet zur Kommunikation mehrere http-Schnittstellen und auch einige Laufzeit-Schnittstellen (z.B. IRC)

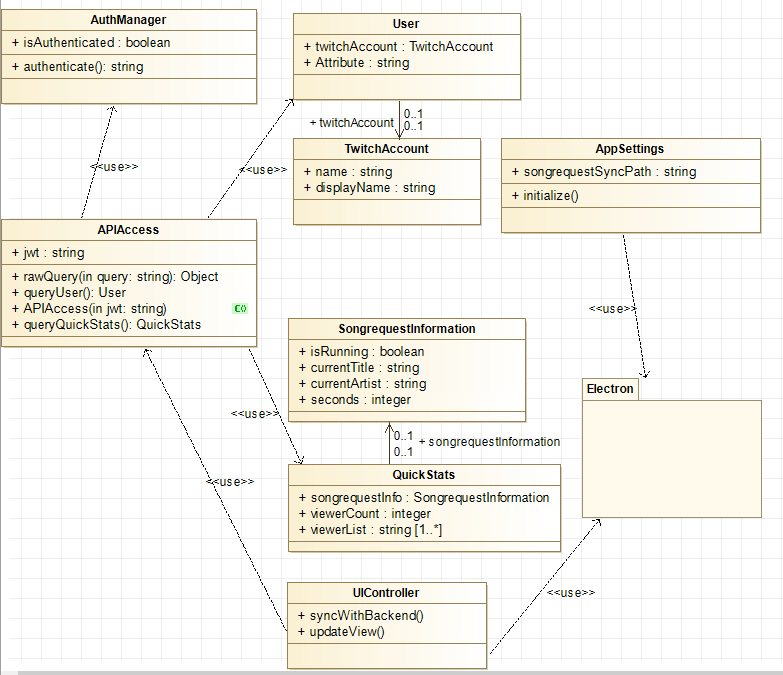


Abbildung 4: Klassendiagramm der Gesamtapplikation

#### 10.1.3 Dynamik

Da es sich nur um eine Applikation handelt, die lediglich mit einer anderen Schnittstelle kommuniziert, und keine weiteren aktiven Komponenten (z.B. Datenbank) hat gibt es hier keinen Bedarf für ein zweites Diagramm.

Die komplexeste Userstory von Twasi-Desktop ist die Authentifizierung, da dabei unterschiedliche Komponenten mitspielen müssen.

Eine kurze Beschreibung der verschiedenen Komponente, die im Sequenzdiagramm dargestellt sind:

*Twasi Desktop Main*

Dies bezeichnet den Main-Prozess, welcher beim Applikationsstart ausgeführt wird. Er integriert sich hierbei tief ins eingesetzte Framework «Electron» und gibt an, wie die Applikation dargestellt werden soll. Weiter verwaltet diese Komponente die geöffneten Fenster und stellt generell die Schnittstelle zur nativen Plattform (Betriebssystem, Dateisystem, etc.) dar.

*Twasi Authentication (OAuth)*

Twasi Authentication ist eine Schnittstelle, welche eigentlich einen Wrapper über die nächste Komponente (Twitch Authentication) darstellt. Hierbei fügt sie jedoch einige spezifische Daten aus und fügt eine Authentifizierungszeichenfolge hinzu. Dies abstrahiert die Schnittstelle zwischen Twasi Desktop und Twitch. So greift Twasi Desktop nie direkt auf Twitch zu, sondern nutzt immer die Zwischenschnittstelle, welche nach Bedarf weitere Daten von Twitch abfragt.

*Twitch Authentication (OAuth)*

Dies stellt den öffentlichen OAuth-Service von Twitch dar, der anderen Applikationen erlaubt, eine Option «Anmelden mit Twitch» hinzuzufügen. Dabei stehen nach dem Anmelden der Applikation diverse Userbezogene Daten wie Email-Adresse, Benutzername oder Profilbild zur Verfügung.

*Twasi Desktop Render*

Dies ist die Komponente, welche innerhalb des Fensters angezeigt wird und verantwortlich ist für die Darstellung der Daten. Diese ist Nahe neben der Twasi Desktop Main-Komponente angesiedelt und führt im späteren Verlauf Kommunikationen mit dieser, da auf Useraktion (z.B. Buttonklick) etwas auf dem nativen System unternommen werden muss (z.B. Datei anlegen).

### 10.2 Schnittstellendefinitionen

Da die Applikation eigenständig keine Daten verwaltet, erfolgt die ganze Kommunikation über eine HTTPS-API. Hierbei wird dabei das Schnittstellen-Framework «GraphQL» verwendet. Da direkt in diesem eine Schnittstellendefinition von möglichen Abfragen enthalten ist, ist eine Dokumentation hier nicht sinnvoll.

Die Dokumentation der GraphQL-API kann entweder unter https://panel-beta.twasi.net/dev oder in Form einer GraphQL-Schemadatei angeschaut werden: https://github.com/Twasi/twasi-core/blob/master/src/main/resources/schema.graphqls

Alle Antworten der API sind dem JSON-Standard entsprechend. Jegliche Anfragen werden serverseitig validiert und mit einer entsprechenden Fehlermeldung abgelehnt, insofern dies notwendig ist.

Die Kommunikation mit der API soll ausdrücklich über das HTTPS-Protokoll erfolgen. Die Schnittstelle der Test-Instanz befindet sich hierbei unter https://api-beta.twasi.net/.

### 10.3 Datenmodel

Da die Datenbank nicht Bestandteil des Twasi-Desktop Projektes ist, sondern nur eine Kommunikation per API unternommen wird, hat dieses Projekt keine Datenbank. Somit liegt kein Entitätsdiagramm vor.

### 10.4 Sicherheit

Da sämtliche Datenhaltung und Verwaltung von der Hauptapplikation Twasi übernommen wird, sind keine zusätzlichen Datenschutzanforderungen einzuhalten.

Damit der Zugriff auf die Daten so sicher wie möglich verläuft, erfolgt sämtliche Kommunikation mit der API über ein verschlüsseltes Protokoll, welches von der Öffentlichkeit als sicher angesehen wird. Hierbei wird hauptsächlich HTTPS (Hyper Text Transfer Protocol Secure) verwendet. Als Verschlüsselungsalgorithmus schreibt der Web-Server Twasi.net mindestens SHA512 oder AES256 vor.

Da die Desktop-Applikation die Authentifizierung grundsätzlich auslagert, respektive an ein bereits bestehendes Backend abgibt, hat die Applikation keine besonders schützenswerten Daten zu behandeln, mit der Ausnahme einer Zeichenreihenfolge, die das Backend zur Identifizierung nutzt (später genannt: Zugriffsschlüssel).

Dieser Schlüssel darf dabei nicht in die Hände von dritten gelangen, da er:

- Freien Zugriff auf alle Dienste von Twasi.net erlaubt (API)

- Jemanden den ursprünglichen Besitzer des Schlüssels ihn imitieren lässt

- Abfrage von Benutzerspezifischen Daten (z.B. Email-Adresse) erlaubt

Mit dem Schlüssel kann jedoch nicht:

- Das Passwort im Klartext ausgelesen werden

- Das Passwort geändert werden

- Eine kritische Änderung am Account durchgeführt werden, wie zum Beispiel die Löschung oder die Änderung der Email-Adresse

Weiter hat ein Schlüssel folgende Eigenschaften:

- Kann jederzeit wiederrufen, bzw. als ungültig erklärt werden

Vorkehrungen zum Schutz des Schlüssels von Drittpersonen:

- Die Kommunikation mit der Backend-API läuft ausschliesslich über HTTPS, da dort der Schlüssel im Klartext mitübermittelt wird. Damit wird er in öffentlichen Netzwerken vor MITM-Attacken (Man-in-the-middle) geschützt.

- Der Schlüssel wird in der Applikation nur jeweils im Arbeitsspeicher und nie auf der Hard Drive gespeichert. Beim Neustart wird ein neuer Schlüssel vom Server angefordert.

## Testkonzept und Testspezifikationen

### 11.1 Unit Test

Unit Tests auf Basis JavaScript sowie Jest.js als Testframework und Runner.

Die Tests einer komplexen Komponente wurden jeweils in derselben Namensgebung der bestehenden JavaScript-Datei gehalten und ein “.test.” eingefügt.

Ein Beispiel für die Komponente “AuthManager.js” kann im Anhang B gefunden werden.

### 11.2 Systemtest - Testspezifikation

#### 11.2.1 Kritikalität der Funktionseinheit

Die wichtigste Komponente von Twasi Desktop stellt die Verbindung zur API dar, wenn diese wegfällt, wird der Rest der Applikation überflüssig und somit hinfällig. Dieser Teil weist daher eine hohe Kritikalität auf.  
Weiter weist auch die Authentifizierungskomponente ein hohes Mass an Kritikalität auf, da die Anmeldung bei Twasi zwingend funktionieren muss.

Auch für diese Komponente wurde ein Unit Test erstellt, welcher die Verbindung vom Client zur API testet, dieser wirkt unterstützend und kann nun durch manuelle Tests erweitert werden.

Eher niedrig ist die Kritikalität bei den restlichen Funktionalitäten, da diese jeweils immer nur eine gewisse Funktion der Applikation, und nie (oder nur bedingt) zu einem Totalausfall führen können.

#### 11.2.2 Testanforderungen

Die Tests von Twasi Desktop müssen den folgenden Anforderungen entsprechen:

- Tests werden mit Normal-, Grenz- und fehlerhaften Werten durchgeführt.

- Tests werden auf dem primären Betriebssystem (Windows) durchgeführt.

- Tests werden auf unterschiedlichen Versionen von Betriebssystemen durchgeführt (Windows 7 + 10)

- Für Tests werden mögliche Situationen nachgestellt (andere Systeme fallen aus z. B.)

#### 11.2.3 Testverfahren

Ein Test für Twasi Desktop wird immer in drei Abschnitte unterteilt, dies orientiert sich am Prinzip von «ordnen, handeln und durchsetzen», einem als «AAA-Pattern» bekannten Ansatz (engl. Arrage, Act and Assert).

Dieser beinhaltet folgenden Abschnitte: «Vorbereitung», «Durchführung» sowie «Auswertung».

Die beschriebenen Tests müssen genau nach den definierten Testanforderungen für Betriebssystem, Programmversion und ausgewählten Testdaten umgesetzt werden.

Zur problemfreien Durchführung der Tests werden im Vorfeld gewisse Basisvoraussetzungen definiert, so muss ein Benutzer über ein Konto bei der Livestreaming-Plattform twitch.tv verfügen und zur Funktionsüberprüfung gewisser Funktionalitäten muss ein Stand an Testdaten hinterlegt werden.

Mit folgendem Benutzerkonto werden die Tests durchgeführt:

Benutzername: twasidesktoptest

Passwort: sml12345

Bei der Durchführung der Tests wird auf die Kritikalität der jeweiligen Teilkomponente (siehe Punkt 11.2.1) geachtet und bei Komponenten mit hoher Kritikalität auf eine einwandfreie Durchführung der Tests geachtet.

#### 11.2.4 Testkriterien

Abdeckungsgrad:

Die einwandfreie Funktionalität der API-Verbindung sowie der Authentifizierung (Verfahren des Ein- und Ausloggens) müssen gewährleistet sein, diese müssen sowohl durch Unit- als auch durch Systemtests komplett abgedeckt werden.

Die restlichen Komponenten werden mit ausschlaggebenden Black-Box-Tests getestet.

Checklisten:  
Als Rahmencheckliste für die Ersteinrichtung der Testumgebung können folgende Punkte beachtet werden:

1. Aufsetzen der Testumgebung
2. Überprüfung Testverfahren
3. Installation Twasi Desktop
4. Nutzung und Einhaltung des (unter Punkt 11.2.3) beschriebenen Testverfahrens

Endkriterien:

Bei den kritischen Komponenten API-Verbindung sowie der Authentifizierung kann ein Test als abgeschlossen betrachtet werden, insofern sämtliche Funktionalitäten im gewünschten Masse (siehe Punkt 11.2.4) und ohne Fehler funktionieren und somit «erfolgreich» benutzt werden können.

Bei den restlichen Komponenten sollen die Tests entsprechend der Testfälle ausgeführt werden, hier können Fehler und Probleme auftreten.

Abstütze der Applikation sind stets gesondert zu vermerken.

#### 11.2.5 Testfälle

|  |  |
| --- | --- |
| **Testfall** | **User Story** |
| Ich sehe in der Applikation eine Anmeldemaske und kann mich mithilfe dieser anmelden. Nach der Anmeldung werde ich auf die Hauptseite der Applikation weitergeleitet. | Authentication (OAuth) |
| Als angemeldeter Benutzer in der Applikation sehe ich einen Button, welcher einen Kommando-Export auslöst. Dabei werden sämtliche existenten, persönlichen Kommandos des Twasi-Bots in eine Datei gespeichert, der Speicherort kann via Explorer ausgewählt werden. | Befehls-Export |
| Als angemeldeter Benutzer in der Applikation sehe ich einen Button, welcher einen SongSync aktiviert. Dabei wird der aktuell abgespielte Song des Streams an einen von mir ausgewählten Dateispeicherort abgespeichert. | SongSync |
| Die Applikation bietet eine einheitliche Navigation, jeder Klick innerhalb der Navigation bringt mich logisch zur gewünschten Unterseite. | Navigation |

### 11.3 Testprozedur

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer kann sich anmelden | |
| **Abgedeckte Anwendungsfälle** | Authentication | |
| **Ausgangssituation** | Der Benutzer hat sich bereits einmal direkt bei Twasi.net angemeldet | |
| **Vorbereitungsschritte** | Die Applikation ist auf dem Testgerät installiert | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Applikation öffnen und mit dem oben angegebenen Testbenutzer via Twitch.tv anmelden | | Die Hauptseite der Applikation erscheint |
| 1. Überprüfen, ob im Menu der korrekte Name (twasidesktoptest) angezeigt wird. | | Der korrekte Name wird angezeigt |
| 1. Abmelden über den Logout-Button im Menu | | Erneut erscheint das Login-Fenster, das mich diesmal nach meinen Twitch.tv-Anmeldedaten fragt |
| 1. Die Applikation schliessen | | Die Applikation geht zu |

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer kann im angemeldeten Zustand seine Befehle exportieren | |
| **Abgedeckte Anwendungsfälle** | Befehls-Export, Authentication | |
| **Ausgangssituation** | Es ist mindestens ein Befehl in Twasi erfasst | |
| **Vorbereitungsschritte** | Die Applikation ist auf dem Testgerät installiert | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Applikation öffnen und mit dem oben angegebenen Testbenutzer via Twitch.tv anmelden | | Die Hauptseite der Applikation erscheint |
| 1. In der Sidebar zu „Befehle“ navigieren | | Der Inhalt der rechten Seite wechselt zur „Befehlsseite“ |
| 1. Den Button Export anwählen | | Es öffnet sich ein File Explorer zur Dateiauswahl |
| 1. Folgenden Pfad auswählen / dazu navigieren:   Desktop/export.txt | | Desktop kann gefunden werden, Dateiname kann eingegeben werden |
| 1. Die Dateiauswahl bestätigen | | Der File Explorer schliesst sich, es erscheint ein Dialog des erfolgreichen Exports |
| 1. Die Datei auf dem Desktop überprüfen | | Der Inhalt der Datei auf dem Desktop stimmt mit den aktuell erstellten Befehlen überein. |

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer kann den aktuell wiedergegebenen Songtitel in einer Datei wiederspiegeln lassen | |
| **Abgedeckte Anwendungsfälle** | SongSync, Authentication | |
| **Ausgangssituation** | Der Benutzer hat sich bereits einmal direkt bei Twasi.net angemeldet | |
| **Vorbereitungsschritte** | Die Applikation ist auf dem Testgerät installiert | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Applikation öffnen und mit dem oben angegebenen Testbenutzer via Twitch.tv anmelden | | Die Hauptseite der Applikation erscheint |
| 1. Rechts in der Sidebar zur Seite „SongSync“ navigieren | | Der rechte Inhalt der Seite ändert sich zur „SongSync“-Seite |
| 1. Die Checkbox auf der rechten Seite aktivieren | | Die Checkbox ändert ihre Farbe, der Button wird aktiv. |
| 1. Den Button „Choose File“ anwählen | | Es öffnet sich ein File Explorer, um eine Datei auszuwählen |
| 1. Die Datei „Desktop\song.txt“ auswählen | | Die Änderung der Einstellung wird bestätigt |
| 1. 5 Sekunden warten | | - |
| 1. Den der Datei „Desktop\song.txt“ mit dem aktuell abgespielten Song vergleichen | | Die beiden Informationen stimmen überein. |
| 1. Die Applikation schliessen und erneut öffnen | | Der Benutzer wird automatisch wieder angemeldet und landet auf der Startseite |
| 1. Rechts in der Sidebar zur Seite „SongSync“ wechseln | | In der Seite überprüfen, ob die Einstellungen immer noch vorhanden sind |

|  |  |  |
| --- | --- | --- |
| **Beschreibung** | Der Benutzer kann sich durch die Applikation navigieren | |
| **Abgedeckte Anwendungsfälle** | Navigation, Authentication | |
| **Ausgangssituation** | Der Benutzer hat sich bereits einmal direkt bei Twasi.net angemeldet | |
| **Vorbereitungsschritte** | Die Applikation ist auf dem Testgerät installiert | |
| **Testschritte** | | **Erwartetes Resultat** |
| 1. Applikation öffnen und mit dem oben angegebenen Testbenutzer via Twitch.tv anmelden | | Die Hauptseite der Applikation erscheint |
| 1. In der Sidebar zur Seite „Overview“ navigieren | | Der rechte Inhalt verändert sich nicht, da man sich bereits auf der Hauptseite befindet. |
| 1. In der Sidebar zur Seite „SongSync“ navigieren | | Der rechte Inhalt wechselt zur Seite „SongSync“. |
| 1. Erneut in der Sidebar zur Seite „Overview“ navigieren | | Der rechte Inhalt wechselt zur Seite „Overview“ |
| 1. Abmelden über den Logout-Button im Menu | | Der Benutzer wird abgemeldet, das Login-Fenster erscheint |
| 1. Die Applikation schliessen | | Die Applikation schliesst sich |

## Einführungskonzept

### 12.1 Zusammenfassung

Das Dokument zum Einführungsbericht dient zur Übersicht des Einführungsprozesses. Es erklärt, welche Schritte notwendig sind, um das Projekt Twasi Desktop einzuführen.

### 12.2 Einführungsplan

Das Projekt Twasi Desktop benötigt keine Migration, da es keine eigene Anlegung von Daten beinhaltet.

Twasi Desktop ist Open Source, daher kann es stets weiterentwickelt werden. Dabei muss beachtet werden, dass das Projekt lauffähig bleibt. Da das Projekt von Twasi und Twitch abhängig ist, müssen bei Änderungen ihrerseits auch bei Twasi Desktop beachtet und gegebenenfalls angepasst werden.

Folgende Schritte der Inbetriebsetzung sind notwendig:

1. Bereitstellung der Applikation für die Entwicklergruppe von Twasi & Twasi Desktop
2. Testung der Applikation im produktiven System
3. Alpha Version wird für alle bereitgestellt
4. Allfällige Stabilisierungen werden durchgeführt

Folgende Risiken bestehen:

1. Twasi oder Twitch wird eingestellt: Dies würde bedeuten, dass Twasi Desktop nicht mehr von Nutzen sein kann und eingestellt wird.

### 12.3 Migrationsplan

#### 12.3.1 Migrationsverfahren

Da die Applikation lediglich die Produktpalette von Twasi erweitert, und nicht ein bestehendes Produkt ersetzt ist keine Migration möglich.

#### 12.3.2 Zeitplan für die Migration

Über den Zeitplan der Migration liegt auf Grund ihrer nicht-existenz keine Informationen vor.

#### 12.3.3 Organisation der Migration

Über die Organisation der Migration liegt auf Grund ihrer nicht-existenz keine Informationen vor.

#### 12.4 Ausbildungsplan

Da die Applikation selbsterklärend wirken soll und eine Umfassende Dokumentation eher benachteiligend wäre (Der Benutzer hat nicht Lust, lange Seiten zu lesen) wird auf eine umfassende Dokumentation verzichtet.

Zu einer einfachen Einführung wird später eine einfache Webseite implementiert, welche den Benutzer Schritt für Schritt durch die Hauptarbeitsschritte führt. Auf dieser Webseite sollen viele Bilder und möglichst wenig Text eingesetzt werden. Ziel ist es, dass der Benutzer die Applikation innerhalb von 5 Minuten komplett bedienen kann.

#### 12.6 Akzeptanztest

Das Projektteam ist mit dem generellen Verlauf und des Abgleiches der Realität zur Planung sehr zufrieden. Der Projektplan wurde jede Woche aktualisiert, und mit ihm gleich der Projektfortschritt überwacht und sichergestellt.

Wir haben einen Projektplan gemäss dem gegebenen Format geführt und diesen wöchentlich aktuell gehalten, da dies Teil des Projektplanes war.

## Testprotokoll

Die Spezifikationen zum Test sind unter Punkt 11.3 zu finden.

### 13.1 Abnahme

**Systemtest 2**

Getestete Version: 1.0, Hotfix 1

Tester: Lars Bärtschi; Cyril Heiniger

Datum, Zeit: 11.12.2018, 13.30 – 14.00

Testfall 1: „Der Benutzer kann sich anmelden“

*Gesamter Testfall erfüllt die Anforderungen.*

Testfall 2: „Der Benutzer kann im angemeldeten Zustand seine Befehle exportieren“

*Gesamter Testfall erfüllt die Anforderungen.*

Testfall 3: „Der Benutzer kann den aktuell wiedergegebenen Songtitel in einer Datei wiederspiegeln lassen“

*Gesamter Testfall erfüllt die Anforderungen.*

Testfall 4: „Der Benutzer kann sich durch die Applikation navigieren“

*Gesamter Testfall erfüllt die Anforderungen.*

|  |  |
| --- | --- |
| **Testdatum** | 11.12.2018, 13:30 – 14:00 |
| **Tester** | Lars Bärtschi; Cyril Heiniger |
| **Gesamttestresultat** | þ Abgenommen  ¨ Abgenommen mit Nacharbeiten  ¨ Nicht abgenommen |
| **Nacharbeiten** | Keine von Nöten. |
| **Unterschrift Lieferant** | Lars Bärtschi |
| **Unterschrift Kunde** | Twasi.net |

## Benutzerdokumentation / Anleitungen

### 14.1 Konfigurations-Dokumentation

Aufgrund der kleinen und simplen React-Applikation kann auf eine Serverinstanz verzichtet werden, die Applikation liegt vollständig auf dem Client des Benutzers und kommuniziert via API mit Twasi,

Systemkonfigurationsparameter liegen daher keine vor.

### 14.2 Benutzerhandbuch

#### 14.2.1 Systemübersicht

Das Hauptziel von Twasi Desktop ist es, den bestehenden Twitch-Bot Twasi um eine Desktop-Applikation zu erweitern, welche dem Benutzer (Streamer auf der Live-Streaming-Plattform Twitch) erweiterte Möglichkeiten bietet und die Reichweite des Twitch-Bots auf das Dateisystem erweitert.

Das System ist mit der JavaScript-Softwarebibliothek React geschrieben und verwendet das Framework Electron, womit die Applikation nahtlos als Desktop-Applikation mit Zugriff auf das Dateisystem des Hostrechners benutzt werden kann.

Twasi Desktop bietet dem Streamer die Möglichkeit, Daten des aktuellen Livestreams (personalisierte Befehle, aktuell gespieltes Musikstück) in eine Datei auf seinem Dateisystem zu exportieren. Diese Datei kann im weiteren Verlauf in Drittprogrammen verwendet werden, so z. B., um das aktuell gespielte Musikstück im Steam anzuzeigen.

Diese Daten werden zu keinem Zeitpunkt in der Applikation selbst gespeichert, sondern wandern direkt von der API auf das Dateisystem. Das Login erfolgt direkt über Twitch, personenbezogene Daten wie Passwörter oder E-Mail-Adressen werden nicht in der Applikation gespeichert.

#### 14.2.2 Anwenderfunktionalität

Eine einwandfreie Installation entsprechend Punkt 3.3 wird vorausgesetzt.

1. Anmeldung - Positivfall
2. Der Anwender startet die Applikation über die Startdatei.
3. Der Anwender gibt seine Anmeldedaten für die Plattform Twitch ein oder erstellt sich ein neues Konto in der dafür vorgesehenen Eingabemaske.
4. Bei korrekten Anmeldedaten wird der Benutzer mit Twitch angemeldet und auf die Startseite der Applikation weitergeleitet.
5. Anmeldung - Fehlerfälle
6. Anmeldung mit Twitch nicht möglich: Ungültige Anmeldedaten à Anmeldedaten prüfen und Anmeldevorgang erneut initiieren. Bei Fortbestehen des Fehlers, das Passwort auf Twitch zurücksetzen. Wiederanlauf bei Punkt 2.
7. Die Applikation bleibt auf einem weissen Bild stehen: Der Titel enthält «signing in with twasi» à Die Internetanbindung überprüfen, Proxyeinstellungen überprüfen, Zugriffe sicherstellen auf [*https://api-beta.twasi.net*](https://api-beta.twasi.net) und [*https://id.twitch.tv*](https://id.twitch.tv), hier wird in beiden Fällen ein 404 Fehler erwartet, die Verbindung kann so überprüft werden. Neustart und Wiederanlauf bei Punkt 1.
8. Fehler beim Applikationsstart: «Die DLL-Datei «Dateiname» kann nicht gefunden werden» à Die Applikation via Internet erneut herunterladen und gemäss Punkt 3.3 installieren.
9. Befehlexport – Positivfall
10. Der Anwender befindet sich gemäss Fall 1) auf der Startseite der Applikation.
11. Der Anwender wählt den Menüpunkt «Command Export» aus, er wird auf die Unterseite zum Kommandoexport weitergeleitet.
12. Der Anwender kann über den Button «Export» einen Speicherort auswählen.
13. Im ausgewählten Dateipfad wird ein neues Textdokument erstellt, dem Benutzer wird dies gemeldet.
14. Der Benutzer kann die Datei lokalisieren und ansehen.
15. Befehlexport – Fehlerfälle
16. Die Datei wird nicht erstellt, obwohl das Programm meldet, dass die Aufgabe erfolgreich ausgeführt wurde à Bitte überprüfen, ob die Datei eventuell versteckt wurde. Weiter prüfen, ob es manuell möglich ist, eine Datei an der gewünschten Stelle anzulegen. Speicherplatzkapazität überprüfen.
17. Es erscheint ein Fehler, dass die Datei nicht erstellt werden könne à häufig hat die Applikation nicht genügend Rechte, um die Datei am gewünschten Ort zu erzeugen. Am besten die Freigabeeinstellungen auf der Festplatte überprüfen oder einen anderen Speicherort auswählen. Wir empfehlen ein persönlicher Ordner, wie zum Beispiel „Dokumente“ oder „Desktop“.
18. SongSync – Positivfall
19. Der Anwender befinde sich gemäss Fall 1) auf der Startseite der Applikation.
20. Der Anwender wählt den Menüpunkt «SongSync“ aus.
21. Der Anwender kann die Funktionalität über den Schiebe-Schalter aktivieren resp. Deaktivieren.
22. Der Anwender kann über den Button «Choose» Path“ einen Pfad auswählen, in dem er die Dateien gespeichert haben möchte.
23. Die Dateien werden in einem regelmässigen Intervall von 5 Sekunden aktualisiert.
24. SongSync – Fehlerfälle
25. Die Datei wird nicht erstellt, obwohl das Programm meldet, dass die Aufgabe erfolgreich ausgeführt wurde à Bitte überprüfen, ob die Datei eventuell versteckt wurde. Weiter prüfen, ob es manuell möglich ist, eine Datei an der gewünschten Stelle anzulegen. Die Speicherplatzkapazität ist zu überprüfen.
26. Es erscheint ein Fehler, dass die Datei nicht erstellt werden könne à häufig hat die Applikation nicht genügend Rechte, um die Datei am gewünschten Ort zu erzeugen. Am besten die Freigabeeinstellungen auf der Festplatte überprüfen oder einen anderen Speicherort auswählen. Wir empfehlen ein persönlicher Ordner, wie zum Beispiel «Dokumente» oder «Desktop».
27. Logout – Positivfall
28. Der Anwender befindet sich gemäss Fall 1) auf der Startseite der Applikation.
29. Der Anwender wählt den Menüpunkt «Logout» aus.
30. Die Applikation löscht danach sämtliche lokalen Anmeldedaten über ihn, ihm wird erneut der Login-Bildschirm angezeigt.
31. Logout – Fehlerfälle
32. Der Logout funktioniert nicht wie gewünscht à oft wird dieses Problem durch das Löschen des lokalen Caches gelöst. Unter Windows befindet sich dieser in C:\Users\<user>\AppData\Roaming\twasi-desktop\Cache. Der User soll zur Fehlerbehebung alle Dateien in diesem Verzeichnis löschen.
33. Overview – Positivfall
34. Der Anwender befindet sich auf dem Menüpunkt «Overview».
35. Der Anwender sieht eine Liste. In dieser Liste sind Daten des momentanen Livestreams vorhanden.
36. Ohne weiteres Zutun des Benutzers werden diese Daten alle 5 Sekunden aktualisiert.
37. Overview – Fehlerfälle
38. Die Liste erscheint nur mit leeren Daten à wahrscheinlich hat die Applikation keine Internetverbindung oder der API-Service kann nicht erreicht werden, in diesen Fällen den Benutzen Computer neu starten oder die Applikation im Zweifelsfall gemäss Punkt 3.3 neu installieren.

### 14.3 Integrations- und Installationshandbuch

Installationsanleitung für Windows 7 und 10

Die Installationsarchivdatei wird von gegebener Quelle heruntergeladen. Danach wird sie in dem gewünschten Installationsverzeichnis entpackt. Hierzu eignet sich das Programm „7-Zip“ am besten, es kann jedoch auch ein beliebiger anderer .zip-Archiv-Entpacker verwendet werden.

Die Applikation kann nachfolgend durch Ausführen der „twasi-desktop.exe“-Datei gestartet werden.

Optional: Verknüpfung auf dem Desktop anlegen

Optional kann auf dem Desktop eine Verknüpfung abgelegt werden. Dies erlaubt, die Applikation einfacher und schneller zu starten. Dazu die „twasi-desktop.exe“-Datei rechtsklicken und im Kontextmenü „Senden an“ -> „Desktop (Verknüpfung erstellen) auswählen. Eine entsprechende Verknüpfung sollte auf dem Desktop erscheinen.

Deinstallationsanleitung für Windows 7 und 10

Zur Deinstallation soll zuerst das Ursprüngliche Installationsverzeichnis bereinigt werden. Hierzu erst sicherstellen, dass sich die Applikation in geschlossenem Zustand befindet. Danach das gesamte Verzeichnis löschen. Es verbleiben keine Reste ausserhalb des Installationsverzeichnisses.

Wenn eine Verknüpfung auf dem Desktop erstellt wurde, kann diese nun auch gelöscht werden.

### 14.4 Supporthandbuch

Twasi Desktop wird durch ein kleines Team an Entwicklern vorangetrieben, ein dauerhaft beständiges Supportteam ist nicht vorhanden.

Ein Supporthandbuch ist im Rahmen der vorliegenden Tätigkeiten nicht vorgesehen.

Der Benutzer sowie Fachpersonen können sich am Punkt 3.2.2 «Anwenderfunktionalität» orientieren, da eine grosse Anzahl an Fehler dort, jeweils unter dem Punkt «Fehlerfälle» aufgeführt sind.

## Projekterfahrung

Während der Arbeit am Projekt Twasi Desktop hatten wir die Möglichkeit uns von Beginn bis Ende mit einer kompletten Entwicklungs- und Dokumentationsprozedur einer Applikation zu beschäftigen.

Die Projektarbeit ist der effektiven Abschlussarbeit der Informatiker (IPA) nah angelehnt und dient in der vorliegenden Form als ideale Vorbereitung zu dieser.

Wir hatten sowohl die Möglichkeit als Gruppe eine von uns erdachte und durch verschiedene Varianten abgeglichene Applikation zu realisieren und somit ein breites Spektrum an neuen Technologien und Arbeitsweisen kennenzulernen, als auch diese Arbeit durch verschiedene Phasen zu strukturieren, diese kennenzulernen und die benötigten Anforderungen zielorientiert zu erarbeiten.

Die Gruppe selbst konnte sich so neues Wissen aneignen, sei es zur Anwendung in der alltäglichen Arbeit aber auch als Vorbereitung auf die spätere Berufstätigkeit oder eine Weiterbildung.

Das genaue Arbeiten wurden geschärft und konnte bei allen Mitgliedern der Gruppe, trotz unterschiedlichen Fähigkeitsniveaus noch geschult werden.

Das nächste Mal möchten wir gerne mehr darauf achten, die Arbeiten so aufzuteilen, dass weniger individuelle Gespräche notwendig sind, welche die Produktivität etwas zurückhalten können.

Die eingesetzten Technologien waren im Gegenzug sehr interessant und modern, so sieht man immer mehr grössere Unternehmen auf diese setzen.

Dies begründet unsere Gedanken während dem Variantenvergleich und zeigt uns, dass wir hiermit auf dem wichtigen Pfad waren. Freude und Spass gemacht hat das Zusammenarbeiten im Team und das Arbeiten an einem „richtigen“ Projekt, was später eventuell eingesetzt werden kann und nicht nur ein Übungsprojekt ist, was nach dem Modul nicht mehr angefasst wird.

# Teil 3

## Dokumentation Quellcode

public/index.html

1. <!DOCTYPE html**>**
2. **<html** lang="en"**>**
3. **<head>**
4. **<meta** charset="utf-8"**>**
5. **<meta** name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"**>**
6. **<meta** name="theme-color" content="#000000"**>**
7. <!--
8. manifest.json provides essential metadata about the application used for offline caching
9. --**>**
10. **<link** rel="manifest" href="%PUBLIC\_URL%/manifest.json"**>**
11. **<link** rel="shortcut icon" href="%PUBLIC\_URL%/favicon.ico"**>**
12. <!--
13. PUBLIC\_URL in the tags above will be replaced with the production url while building.
14. --**>**
15. **<title>**Twasi-Desktop**</title>**
16. **</head>**
17. **<body>**
18. **<noscript>**
19. You need to enable JavaScript to run this app.
20. **</noscript>**
21. **<div** id="root"**></div>**
22. <!--
23. This HTML file is the template, anything can be added later on here.
24. --**>**
25. **</body>**
26. **</html>**

public/manifest.json

1. {
2. "short\_name": "Twasi-Desktop",
3. "name": "Twasi-Desktop",
4. "icons": [
5. {
6. "src": "favicon.ico",
7. "sizes": "64x64 32x32 24x24 16x16",
8. "type": "image/x-icon"
9. }
10. ],
11. "start\_url": "./index.html",
12. "display": "standalone",
13. "theme\_color": "#000000",
14. "background\_color": "#ffffff"
15. }

package.json

1. {
2. "name": "twasi-desktop",
3. "version": "0.1.0",
4. "private": **true**,
5. "dependencies": {
6. "@material-ui/core": "^3.4.0",
7. "@material-ui/icons": "^3.0.1",
8. "jwt-decode": "^2.2.0",
9. "react": "^16.5.0",
10. "react-dom": "^16.5.0",
11. "react-scripts": "^2.1.1"
12. },
13. "main": "src/electron-starter.js",
14. "homepage": "./",
15. "scripts": {
16. "start": "cross-env BROWSER=none react-scripts start",
17. "build": "react-scripts build",
18. "test": "react-scripts test --env=jsdom",
19. "test-noni": "cross-env CI=true react-scripts test --env=jsdom",
20. "eject": "react-scripts eject",
21. "electron": "electron .",
22. "electron-dev": "cross-env ELECTRON\_START\_URL=http://localhost:3000 electron .",
23. "dev": "concurrently --kill-others \"npm start\" \"npm run electron-dev\"",
24. "electron-build": "npm run build && npm run electron",
25. "package": "electron-packager . twasi-desktop"
26. },
27. "devDependencies": {
28. "concurrently": "^4.0.1",
29. "cross-env": "^5.2.0",
30. "electron": "^3.0.6",
31. "electron-packager": "^12.2.0",
32. "jest": "^23.6.0",
33. "material-ui": "^0.20.2",
34. "react-redux": "^5.1.0",
35. "react-router-dom": "^4.3.1"
36. },
37. "browserslist": [
38. ">0.2%",
39. "not dead",
40. "not ie <= 11",
41. "not op\_mini all"
42. ]
43. }

src/api/APIConnector.js

1. /\*\*
2. \* APIConnector holds the connection to the GraphQL backend.
3. \*
4. \* Lars Bärtschi, 27.11.2018
5. \*/
7. **import** AuthManager from '../auth/AuthManager';
8. **import** fetch from './fetch';
10. /\*\*
11. \* Returns a formatted GraphQL request using the provided query and the configured GRAPHQL\_URL.
12. \* @param query the query to use
13. \* @returns {Promise<data>}
14. \*/
15. **const** getGraph = query => fetch(window.env.GRAPHQL\_URL, 'post', `query=${encodeURI(query)}`);
17. /\*\*
18. \* Returns a formatted GraphQL request for a certain user
19. \* @param query the query to use
20. \* @param jwt the user's jwt token
21. \* @param target the GraphQL target
22. \* @returns {Promise<data>}
23. \*/
24. **const** getUserGraph = (query, jwt, target) => {
25. **if** (jwt === **null**) {
26. // eslint-disable-next-line
27. console.log('Not authenticated, could not do request.');
28. **return** **null**;
29. }
30. **return** **new** Promise(resolve => {
31. getGraph(`query{${target}(token:"${jwt}"){${query}}}`).then(data => {
32. resolve(data.data[target]);
33. });
34. });
35. };
37. **class** APIConnector {
38. constructor() {
39. **this**.readyHandlers = [];
40. **this**.isReady = **false**;
41. }
43. /\*\*
44. \* Set's up the local settings, calls all callback handlers since auth context is available
45. \* @param jwt
46. \*/
47. setup(jwt) {
48. **this**.jwt = jwt;
49. **this**.isReady = **true**;
51. **this**.readyHandlers.forEach(handler => handler());
52. **this**.readyHandlers = [];
53. }
55. /\*\*
56. \* Does a request to a certain
57. \* @param query the query to execute
58. \* @param target the GraphQL target
59. \* @returns {Promise<data>}
60. \*/
61. doPanelRequest(query, target) {
62. **if** (!target) {
63. target = "panel";
64. }
66. **if** (!**this**.isReady) {
67. console.error("Tried to do panel request but APIConnector is not ready yet.");
68. **return**;
69. }
70. **return** getUserGraph(query, **this**.jwt, target);
71. }
73. /\*\*
74. \* Returns data about the currently signed in user
75. \* @returns {Promise<data>}
76. \*/
77. getUserInfo() {
78. **return** **this**.doPanelRequest('user{id,twitchAccount{email}}');
79. }
81. /\*\*
82. \* Register the provided handler to be called as soon as the auth context is available
83. \* If the auth context is already available while the function is called, the event handler
84. \* will be called immediately.
85. \* @param eventHandler
86. \*/
87. ready(eventHandler) {
88. **if** (**this**.isReady) {
89. eventHandler();
90. } **else** {
91. **this**.readyHandlers.push(eventHandler);
92. }
93. }
94. }
96. **const** connector = **new** APIConnector();
97. AuthManager.registerOnAuthenticate(jwt => connector.setup(jwt));
99. **export** **default** connector;
101. **export** { APIConnector };

src/api/APIConnector.test.js

1. **import** { APIConnector } from './APIConnector'
3. describe('APIConnector', () => {
5. it('runs the callback as soon as authentication info is available', () => {
6. **const** cb = jest.fn();
7. **const** connector = **new** APIConnector();
9. connector.ready(cb);
11. connector.setup("myJwt");
13. expect(cb).toHaveBeenCalled();
14. });
16. });

src/api/fetch.js

1. /\*\*
2. \* Fetch is an easier to use wrapper for the fetch api.
3. \* It is able to add custom headers and handle some custom errors.
4. \*
5. \* Lars Bärtschi, 27.11.2018
6. \*/
8. **const** requestHeaders = () => ({
9. Accept: 'application/json',
10. 'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8'
11. });
13. /\*\*
14. \* Do a request
15. \* @param url the provided url to make a request to
16. \* @param method (for example GET, POST, PUT, DELETE)
17. \* @param payload only if the requested method has a body (POST/PUT/PATCH)
18. \* @returns {Promise<\*>}, data if completed
19. \*/
20. **export** **default** async (url, method, payload) => {
21. **const** options = {
22. method,
23. headers: requestHeaders(),
24. body: method !== 'GET' ? payload : undefined
25. };
27. **const** response = await fetch(url, options);
28. **if** (response.status === 204) {
29. **return** {};
30. }
32. **const** body = await response.json();
34. **if** (response.status >= 300) {
35. **const** error = **new** Error(body.message);
36. error.response = body;
37. error.status = response.status;
38. **throw** error;
39. }
41. **return** body;
42. };

src/auth/AuthManager.js

1. /\*\*
2. \* AuthManager holds the current state and information of the authentication.
3. \* It allows for other components of the application to register and wait for the
4. \* event of a successful authentication
5. \*
6. \* Lars Bärtschi, 27.11.2018
7. \*/
9. **import** jwtdecode from 'jwt-decode';
11. **if** (!window.signin) {
12. //console.log('Waiting for JWT...');
13. window.signin = jwt => {
14. manager.authenticate(jwt)
15. }
16. }
18. **class** AuthManager {
19. constructor() {
20. **this**.isAuthenticatedValue = **false**;
21. **this**.eventHandlers = [];
22. }
24. /\*\*
25. \* Returns if the user is authenticated
26. \* @returns {boolean}
27. \*/
28. isAuthenticated() {
29. **return** **this**.isAuthenticatedValue;
30. }
32. /\*\*
33. \* Authenticate a user using the provided JWT.
34. \* @param jwt
35. \*/
36. authenticate(jwt) {
37. **this**.jwt = jwt;
38. **this**.userByJwt = jwtdecode(jwt);
40. **this**.isAuthenticatedValue = **true**;
42. **this**.eventHandlers.forEach(handler => handler(jwt));
43. }
45. /\*\*
46. \* Registers an event handler that gets called as soon as authentication
47. \* has completed.
48. \* @param eventHandler
49. \*/
50. registerOnAuthenticate(eventHandler) {
51. **this**.eventHandlers.push(eventHandler);
52. }
53. }
55. **const** manager = **new** AuthManager();
57. **export** **default** manager;
59. **export** { AuthManager };

src/auth/AuthManager.test.js

1. **import** { AuthManager } from './AuthManager';
3. describe('AuthManager', () => {
5. it('is not authenticated by default', () => {
6. **const** auth = **new** AuthManager();
8. expect(auth.isAuthenticated).toBe(**false**);
9. });
11. it('runs authentication callback after authentication info is available', () => {
12. **const** cb = jest.fn();
13. **const** auth = **new** AuthManager();
15. auth.registerOnAuthenticate(cb);
17. auth.authenticate("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c");
19. expect(cb).toHaveBeenCalled();
20. });
22. it('decodes the token correctly', () => {
23. **const** auth = **new** AuthManager();
25. auth.authenticate("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c");
27. expect(auth.userByJwt.name).toBe("John Doe");
28. });
30. });

src/components/header/Header.css

1. .header {
2. **background-color**: transparent;
3. **border-top**: #00aeae 5px solid;
4. **height**: 150px;
5. }
7. #username {
8. **color**: white;
9. **position**: fixed;
10. top: 5px;
11. **font-size**: 1.2em;
12. **margin-left**: calc(15% - 95px);
13. }

src/components/header/Header.js

1. /\*\*
2. \* Header is the visible component on top of the application, which shows the current logged in user.
3. \*
4. \* Marco Steiner, 04.12.2018
5. \*/
7. **import** React, { Component } from 'react';
8. **import** './Header.css';
9. **import** APIConnector from '../../api/APIConnector';
11. **class** Header **extends** Component {
13. constructor(props) {
14. **super**(props)
15. **this**.state = {
16. username: "checking..."
17. }
19. **this**.getUserName()
20. }
22. getUserName() {
23. APIConnector.ready(() => {
24. APIConnector.getUserInfo().then((x) => {
25. **this**.state.username = x.user.twitchAccount.email
26. **this**.setState(**this**.state);
27. });
28. });
29. }
31. render() {
32. **return** (
33. <div className="background">
34. <div className="header" />
35. <p id="username">{  **this**.state.username  }</p>
36. </div>
37. );
38. }
39. }
41. **export** **default** Header;

src/components/info-tiles/InfoTiles.js

1. /\*\*
2. \* InfoTile holds the overview of the page: It is highly customizable.
3. \*
4. \* Marco Steiner, 27.11.2018
5. \*/
7. **import** React, { Component } from 'react';
8. **import** APIConnector from '../../api/APIConnector';
9. **import** Paper from '@material-ui/core/Paper';
10. **import** List from '@material-ui/core/List';
11. **import** ListItem from '@material-ui/core/ListItem';
13. **import** { getListStyle, getItemStyle, getTitleStyle } from './\_style';
15. **import** './\_style.css';
17. **class** InfoTiles **extends** Component {
18. constructor(props) {
19. **super**(props);
20. **this**.state = {
21. items : [
22. {
23. title: 'Requested Song',
24. query: 'current/request/song/name',
25. target: 'songrequest',
26. id: 'overviewSongName',
27. value: ""
28. },
29. {
30. title: 'Song Status',
31. query: 'current/status',
32. target: 'songrequest',
33. id: 'overviewSongStatus',
34. value: ""
35. }
36. ]
37. }
38. }
40. componentDidMount() {
41. **this**.state.items.forEach(e => {
42. **this**.getInfo(e)
43. });
44. }
46. getInfo(item) {
47. **var** query = "";
48. **var** queryEnd = "";
49. **var** parts = item["query"].split("/");
51. parts.forEach(part => {
52. queryEnd += "}";
53. query += part + "{";
54. });
56. query = query.slice(0, -1) + queryEnd.slice(0, -1);
58. APIConnector.ready(() => {
59. APIConnector.doPanelRequest(query, item.target).then(result => {
60. **var** value = result
61. parts.forEach(part => {
62. **if**(!value[part])
63. **return**
64. value = value[part]
65. });
66. **const** newItem = {...item, value}
68. **this**.setState({items:[...**this**.state.items.filter(i => i.id !== item.id), newItem]})
69. });
70. });
72. }
74. render() {
75. **const** renderItems = () =>
76. **this**.state.items.map(item => (
77. <ListItem
78. key={item.id}
79. style={getItemStyle()}
80. innerDivStyle={{ padding: '0px 16px 0px 52px' }}
81. >
82. { item.title }:
83. <i className="tileValue">{ item.value }</i>
84. </ListItem>
85. ));
87. **return** (
88. <Paper style={getListStyle()}>
89. <div style={getTitleStyle()}>
90. Current Stream
91. </div>
92. <List className="tileList">
93. {renderItems()}
94. </List>
95. </Paper>
96. );
97. }
98. }
100. **export** **default** InfoTiles;

src/components/info-tiles/\_style.css

1. .tileValue {
2. **position**: absolute;
3. right: 15px;
4. **max-width**: 60%;
5. **line-height**: 18px;
6. }
8. .tileList {
9. **padding**: 0 !important;
10. }

src/components/sidebar/Sidebar.js

1. /\*\*
2. \* The sidebar component is the visible navigation on the left of the content.
3. \* Pages can be added with ease.
4. \*
5. \* Marco Steiner, 27.11.2018
6. \*/
8. **import** React, { Component } from 'react';
9. **import** PropTypes from 'prop-types';
10. **import** { withRouter } from 'react-router-dom';
11. **import** find from 'lodash/fp/find';
12. **import** { throttle } from 'lodash';
13. **import** Paper from '@material-ui/core/Paper';
14. **import** MenuList from '@material-ui/core/MenuList';
15. **import** MenuItem from '@material-ui/core/MenuItem';
17. **import** { getMenuStyle, getHeaderMenuItem } from './\_style';
19. **import** './\_style.css';
21. **class** Sidebar **extends** Component {
22. constructor(props) {
23. **super**(props);
25. **this**.items = [
26. {
27. key: 'overview',
28. path: '/',
29. icon: 'home',
30. name: 'Overview'
31. },
32. {
33. key: 'commandexport',
34. path: '/export',
35. icon: 'code',
36. name: 'Befehle'
37. },
38. {
39. key: 'songsync',
40. path: '/songsync',
41. icon: 'library\_music',
42. name: 'SongSync'
43. },
44. {
45. key: 'logout',
46. path: '/logout',
47. icon: 'keyboard\_return',
48. name: 'Logout'
49. }
50. ];
52. **this**.handleClick = **this**.handleClick.bind(**this**);
54. **this**.resetAnimation = throttle(
55. () => {
56. **this**.LogoDOM.src = **this**.Logo.src;
57. },
58. 2500,
59. { trailing: **false** }
60. );
61. }
63. handleClick(event, value) {
64. **const** { history } = **this**.props;
65. **const** { path } = find(newItem => newItem.key === value, **this**.items);
66. history.push(path);
67. **this**.setState({});
68. }
70. render() {
71. let { location } = **this**.props
72. let selectedKey = find(item => item.path === location.pathname, **this**.items);
73. **if** (**typeof** selectedKey === 'undefined') {
74. selectedKey = **this**.items[0].key;
75. } **else** {
76. selectedKey = selectedKey.key;
77. }
78. **const** renderItems = () =>
79. **this**.items.map(item => (
80. <MenuItem
81. style={{ fontSize: 13 }}
82. value={item.key}
83. key={item.key}
84. selected={item.key === selectedKey}
85. innerDivStyle={{ padding: '0px 16px 0px 52px' }}
86. onClick={event => **this**.handleClick(event, item.key)}
87. >
88. <i className="material-icons" style={{ marginRight: '15px' }}>{item.icon}</i>
89. { item.name }
90. </MenuItem>
91. ));
93. **return** (
94. <Paper style={getMenuStyle()} className="sidebar">
95. <div style={getHeaderMenuItem()}>
96. Navigation
97. </div>
98. <MenuList
99. className="Sidebar"
100. >
101. {renderItems()}
102. </MenuList>
103. </Paper>
104. );
105. }
106. }
108. Sidebar.propTypes = {
109. location: PropTypes.shape({
110. pathname: PropTypes.string.isRequired
111. }).isRequired,
112. history: PropTypes.shape({
113. push: PropTypes.func.isRequired
114. }).isRequired,
115. userName: PropTypes.string
116. };
118. **export** **default** withRouter(Sidebar);

src/components/sidebar/\_style.css

1. .Sidebar {
2. **padding**: 0 !important;
3. }

src/sites/CommandExport.js

1. /\*\*
2. \* CommandExport component holds the page content and logic of the
3. \* command export page.
4. \*
5. \* Yvo Hofer, 27.11.2018
6. \*/
8. **import** React, {Component} from 'react';
9. **import** APIConnector from '../api/APIConnector';
11. **class** CommandExport **extends** Component {
13. constructor(props) {
14. **super**(props);
15. **this**.remote = window.require("electron").remote;
16. }
18. exportCommands() {
19. APIConnector.ready(() => {
21. APIConnector.doPanelRequest("commands{id, name, content}", "commands")
22. .then(result => {
24. **const** app = window.require('electron').remote;
25. **const** dialog = app.dialog;
26. **const** fs = window.require('fs');
28. dialog.showSaveDialog({
29. filters: [{
30. name: 'Text Documents (\*.txt)',
31. extensions: ['txt']
32. }]
33. }, (fileName) => {
34. **if** (fileName === undefined) {
35. console.log("You didn't save the file");
36. **return**;
37. }
39. fs.writeFile(fileName, JSON.stringify(result), (err) => {
40. **if** (err) {
41. alert("An error occurred creating the file " + err.message)
42. }
44. alert("The file has been successfully exported");
46. });
47. });
48. })
49. });
51. }
53. render() {
54. **return** (
55. <div>
56. <p>Hey, here you're able to **export** your custom commands. Have fun!</p>
57. <button className="exportButton" onClick={**this**.exportCommands}>Export</button>
58. </div>
59. );
60. }
61. }
62. **export** **default** CommandExport;

src/sites/Error404.js

1. /\*\*
2. \* Error404 component is displayed if a page is request that does not exist or is not registered.
3. \*
4. \* Marco Steiner, 27.11.2018
5. \*/
7. **import** React, { Component } from 'react';
9. **class** Overview **extends** Component {
10. render() {
11. **return** (
12. <p>Oh no, we can't find you requested page :/</p>
13. );
14. }
15. }
16. **export** **default** Overview;

src/sites/Logout.js

1. /\*\*
2. \* This logout components logs you out once it would become visible on screen.
3. \* It makes the connection to electron and emits the logout event.
4. \*
5. \* Lars Bärtschi, 27.11.2018
6. \*/
8. **import** React, { Component } from 'react';
10. **class** Logout **extends** Component {
11. componentDidMount() {
12. **this**.remote = window.require('electron').remote;
13. **this**.remote.app.emit("logout");
14. }
16. render() {
17. **return** (
18. <p>Logout/</p>
19. );
20. }
21. }
23. **export** **default** Logout;

src/sites/Overview.js

1. /\*\*
2. \* Overview is the page that is displayed under the overview entry in the sidebar.
3. \* It returns the InfoTiles, which can show general stats.
4. \*
5. \* Marco Steiner, 27.11.2018
6. \*/
8. **import** React, { Component } from 'react';
9. **import** InfoTiles from '../components/info-tiles/InfoTiles';
11. **class** Overview **extends** Component {
12. render() {
13. **return** (
14. <div>
15. <p className="siteTitle">Overview</p>
16. <div className="tiles">
17. <InfoTiles />
18. </div>
19. </div>
20. );
21. }
22. }
24. **export** **default** Overview;

src/sites/SongSync.js

1. /\*\*
2. \* The SongSync component manages the view and logic of the SongSync functionality.
3. \*
4. \* Lars Bärtschi, 27.11.2018
5. \*/
7. **import** React, {Component} from 'react';
8. **import** APIConnector from '../api/APIConnector';
9. **import** Switch from '@material-ui/core/Switch';
11. **class** SongSync **extends** Component {
13. constructor(props) {
14. **super**(props);
16. **this**.state = {isDisabled: **true**};
18. **this**.toggleButton = **this**.toggleButton.bind(**this**);
19. **this**.songSync = **this**.songSync.bind(**this**);
20. }
22. toggleButton(e, checked) {
23. **this**.setState({isDisabled: !checked});
24. }
26. songSync() {
27. **const** component = **this**;
28. **const** app = window.require('electron').remote;
29. **const** fs = window.require('fs');
30. **const** dialog = app.dialog;
32. **function** updateFile(folderPath) {
33. APIConnector.ready(() => {
34. APIConnector.doPanelRequest("current{status, secondsIn, request{song{name}}}", "songrequest")
35. .then(result => {
36. let commandToExport = result.current.request.song.name;
37. fs.writeFile(folderPath[0] + '\\SongExport.txt', commandToExport, (err) => {
38. **if** (err) {
39. alert("An error occurred creating the file " + err.message)
40. }
41. console.log(component);
42. **if** (!component.state.isDisabled) {
43. setTimeout(() => updateFile(folderPath), 5000);
44. }
45. });
46. })
47. });
48. }
50. dialog.showOpenDialog({
51. properties: ['openDirectory']
52. }, **function** (folderPath) {
53. **if** (folderPath === undefined) {
54. console.log("You didn't choose a folder");
55. } **else** {
56. updateFile(folderPath);
57. setTimeout(() => updateFile(folderPath), 5000);
58. }
59. });
60. }
62. render() {
63. **return** (
64. <div>
65. <p>Welcome to SongSync</p>
66. <Switch onChange={**this**.toggleButton} className="syncSwitch"/>
67. <button onClick={**this**.songSync} className={"syncButton"} disabled={**this**.state.isDisabled}>choose path
68. </button>
69. </div>
70. );
71. }
72. }
74. **export** **default** SongSync;

src/App.js

1. /\*\*
2. \* App is the main component of the application.
3. \* It handels the routing and decides which page to render where.
4. \*
5. \* Lars Bärtschi, 27.11.2018
6. \*/
8. **import** React, {Component} from 'react';
9. **import** { BrowserRouter, Route, Switch } from 'react-router-dom';
11. **import** Header from './components/header/Header';
12. **import** Sidebar from './components/sidebar/Sidebar';
13. **import** Overview from './sites/Overview';
14. **import** Logout from './sites/Logout';
15. **import** Error404 from './sites/Error404';
16. **import** CommandExport from './sites/CommandExport';
18. **import** './App.css';
19. **import** SongSync from "./sites/SongSync";
21. **class** App **extends** Component {
22. render() {
23. **return** (
24. <BrowserRouter>
25. <div>
26. <Header/>
27. <Sidebar/>
28. <div className="App-content">
29. <Switch>
30. <Route exact path='/' component={Overview}/>
31. <Route exact path='/logout' component={Logout}/>
32. <Route exact path='/export' component={CommandExport}/>
33. <Route exact path='/songsync' component={SongSync}/>
34. <Route path='/' component={Error404}/>
35. </Switch>
36. </div>
37. </div>
38. </BrowserRouter>
39. );
40. }
41. }
43. **export** **default** App;

src/electron-startet.js

1. /\*\*
2. \* Electron-Starter is the entry point of the application.
3. \*
4. \* It handels several events, like state changes, initiates authentication and also handels
5. \* connections to native API's
6. \*
7. \* Lars Bärtschi, 27.11.2018
8. \*/
10. **const** electron = require('electron');
11. **const** app = electron.app;
12. **const** BrowserWindow = electron.BrowserWindow;
14. **function** createWindow() {
15. // Create the browser window.
16. app.mainWindow = **new** BrowserWindow({width: 800, height: 600});
18. // and load the index.html of the app.
19. **const** startUrl = process.env.ELECTRON\_START\_URL || `file://${\_\_dirname}/../build/index.html`;
20. app.mainWindow.loadURL(startUrl);
22. app.mainWindow.on('closed', **function** () {
23. app.mainWindow = **null**
24. })
25. }
27. // Quit when all windows are closed.
28. app.on('window-all-closed', **function** () {
29. // On OS X it is common for applications and their menu bar
30. // to stay active until the user quits explicitly with Cmd + Q
31. **if** (process.platform !== 'darwin') {
32. app.quit()
33. }
34. });
36. app.on('activate', **function** () {
37. // On OS X it's common to re-create a window in the app when the
38. // dock icon is clicked and there are no other windows open.
39. **if** (app.mainWindow === **null**) {
40. createWindow()
41. }
42. });
44. // Log the user out, clear browser history and cache
45. app.on('logout', **function**() {
46. console.log('Logging out...');
47. app.mainWindow.webContents.session.clearCache(() => {});
48. app.mainWindow.webContents.session.clearStorageData();
49. app.mainWindow.webContents.clearHistory();
51. app.emit('ready');
53. app.mainWindow.close();
54. });
56. // If the app content is loaded, start the authentication
57. app.on('ready', () => {
58. app.authWindow = **new** BrowserWindow({
59. width: 800,
60. height: 600,
61. alwaysOnTop: **true**,
62. autoHideMenuBar: **true**,
63. webPreferences: {
64. nodeIntegration: **false**
65. }
66. });
68. app.authWindow.setTitle("Signing in with Twasi.net...");
70. // After the user gets redirected to the frontend, close the app window and fetch the JWT token
71. app.authWindow.webContents.on('did-navigate', **function** (event, newUrl) {
72. **const** beginning = "https://panel-beta.twasi.net/?jwt=";
74. **if** (newUrl.startsWith(beginning)) {
75. **const** jwt = newUrl.replace(beginning, '');
76. createWindow();
78. app.mainWindow.webContents.on('did-finish-load', **function**() {
79. app.mainWindow.webContents.executeJavaScript("window.signin('" + jwt + "')");
80. });
82. app.authWindow.close();
83. }
84. });
86. // Open OAuth provider in browser
87. app.authWindow.loadURL('https://api-beta.twasi.net/auth?environment=https://panel-beta.twasi.net');
88. });

 src/index.js

1. /\*\*
2. \* This is the main entry point of the rendered application.
3. \*
4. \* It just renders the component App, which handels everything further.
5. \*
6. \* Lars Bärtschi, 27.11.2018
7. \*/
9. **import** React from 'react';
10. **import** ReactDOM from 'react-dom';
11. **import** './index.css';
12. **import** App from './App';
13. **import** registerServiceWorker from './registerServiceWorker';
15. **import** './config';
16. **import** './api/APIConnector';
17. **import** './auth/AuthManager';
19. ReactDOM.render(<App />, document.getElementById('root'));
20. registerServiceWorker();

src/config.js

1. /\*\*
2. \* Config.js holds some general configuration about the application.
3. \*/
5. window.env = {
6. ...window.env,
7. GRAPHQL\_URL: "https://api-beta.twasi.net/graphql"
8. };

## Literaturverzeichnis

Dokumentation zum Framework Electron: <https://electronjs.org/docs>.

Dokumentation zum Framework React: <https://reactjs.org/docs/getting-started.html>.

Dokumentation zum Twitch Chatbot Twasi: <https://docs.twasi.net/en/>.

Dokumentation zur API der Streamingplattform Twitch: <https://dev.twitch.tv/docs/>.

## Glossar

**Twitch**

Twitch, auch Twitch.tv, ist ein Live-Streaming-Videoportal, das vorrangig zur Übertragung von Videospielen genutzt wird.

**Livestream**

Als Livestream oder Live-Streaming, zu Deutsch Echtzeitübertragung, bezeichnet man ein Streaming-Media-Angebot (Video oder Audio), das in Echtzeit (englisch live) bereitgestellt wird.

In unserem Kontext wird als Plattform das Live-Streaming-Videoportal Twitch genutzt.

**Streamer**

Person oder Personen, welche den Bildschirminhalt ihres Computers (Gaming-Livestream), sich selbst (Community-Livestream) oder die Videoaufnahme einer Kamera (Realwelt-Livestream) live, also in Echtzeit auf einer dafür geeigneten Live-Streaming-Videoplattform zur Verfügung stellen.

**Bot**

Ein Bot ist eine Abkürzung für das Wort Robot und stellt, in unserem Kontext, einen automatisierten Chat-Teilnehmer dar.

Dieser Unterstützt den Zuschauer eines Livestreams mit Informationen und verhindert Spam.

**Twasi**

Akronym; Twitch Assistant

Ist eine frei nutz-, und erweiterbare Anwendung, welche einen Chatbot für das Live-Streaming-Videoportal Twitch zur Verfügung stellt.

**OBS**

Open Broadcaster Software ist eine freie Software, um den Bildschirminhalt eines Computers (Spiele, Videokameras) aufzuzeichnen.